实验 1.1 套接字基础与 UDP 通信

一． 实验目的

熟悉基于 Python 进行 UDP 套接字编程的基础知识，掌握使用 UDP 套接字发送和接收数据包，以及设置正确的套接字超时，了解 Ping 应用程序的基本概念，并理解其在简单判断网络状态，例如计算数据包丢失率等统计数据方面的意义。

二． 实验内容

1. 操作系统附带的标准 Ping 命令使用 ICMP 进行通信，本实验要求学生编程实现一个简单的，非标准的，基于 UDP 进行通信的 Ping 程序。学生需要用 Python 编写一个 Ping 客户端。客户端程序发送一个 ping 报文，然后接收一个从已经提供的服务器上返回的对应 pong 报文，并计算出从该客户发送 ping 报文到接收到 pong 报文为止的往返时延（Round-Trip Time，RTT）。
2. 在客户端程序一次执行过程中，学生编写的的 Ping 客户端程序需经 UDP 向服务器发送 10 个 ping 报文。对于每个报文，当对应的 pong 报文返回时，客户端程序要确认并打印输出 RTT 值；在整个执行过程中，客户端程序需要考虑分组丢失情况，客户端最多等待 1 秒，超过该时长则打印丢失报文。

三． 实验原理、方法和手段

UDP 作为一种传输层协议，只提供了无连接通信，且不对传送的数据包进行可靠性保证，因此只适合于一次传输少量数据的应用场景，如果在传输过程中需要保证可靠性，则这种可靠性应该由应用层负责。本实验创建的 Ping 程序正是一种不需要保证可靠性的程序，并需要利用这种不可靠性来测量网络的联通情况。

虽然 UDP 不保证通信的可靠性，包到达的顺序，也不提供流量控制。但正是因为 UDP 的控制选项较少，所以在数据传输过程中延迟小、数据传输效率高，一些对可靠性要求不高，但对性能等开销更敏感的应用层协议会选择基于 UDP 进行实现，常见的使用 UDP 的应用层协议包括 TFTP、SNMP、NFS、DNS、BOOTP 等，通常占用 53（DNS）、69（TFTP）、161（SNMP）等端口。

基于 UDP 的无连接客户/服务器在 Python 实现中的工作流程如下：

1. 首先在服务器端通过调用s o c k e t ( ) 创建套接字来启动一个服务器；
2. 服务器调用 bind( ) 指定服务器的套接字地址，然后调用 等待接收数据。
3. 在客户端调用 s o c k e t ( ) 创建套接字，然后调用 s e n d t o ( ) 向服务器发送数据。
4. 服务器接收到客户端发来的数据后，调用 s e n d t o ( ) 向客户发送应答数据，

5. 客户调用 recvfrom() 接收服务器发来的应答数据。

6. 一旦数据传输结束，服务器和客户通过调用 close() 来关闭套接字。

注意在不同的计算机语言实现中，上述调用的名字和具体工作流程可能略有不同。基于 Python 的 UDP 程序工作详细流程如图1.1–1所示。

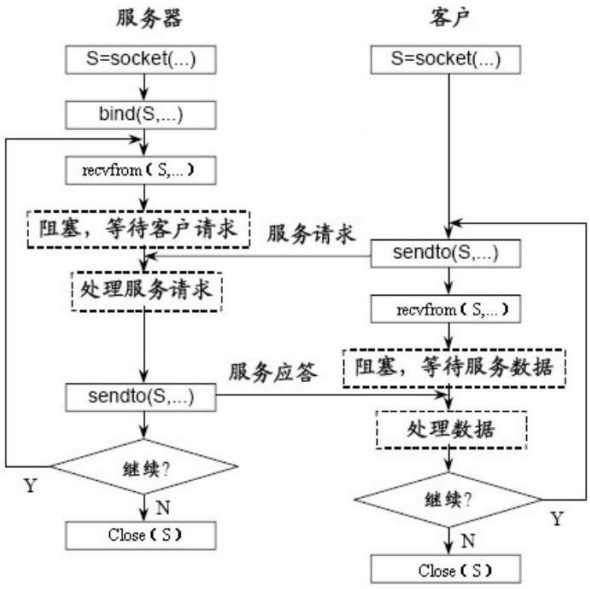

图 1.1–1 无连接客户/服务器流程图

基于 Python 进行 UDP 消息的接收操作时，Python 程序将工作在阻塞状态，即未收到数据包时，Python 程序将挂起等待而不会继续执行。如果程序运行中网络连接出现了问题，导致数据包无法及时到达，这种阻塞式的工作模式将会严重的干扰程序的执行。为了解决这个问题，Python 的套接字通信库提供了一种“超时”机制来防止程序卡死。在 Python 套接字程序中，套接字对象提供了一个 Settimeout()方法来限制 recvfrom() 函数的等待时间，当 recvfrom() 函数阻塞等待超过这个时间（一般称为“超时时间”）后仍然没有收到数据时，程序将会抛出一个异常来说明发生了等待数据接收超时事件。在编写 Python 网络通信程序时，可以利用这个机制来判断是否接收数据超时。

四． 实验条件

- 装有 python 环境的电脑两台；
- 局域网环境；
- 服务器程序（附件中已给出）；
- Python 语言参考手册 – UDP 部分[1]。

[1]可以参考 Python3 官方手册的 套接字部分，也可以查询其他相关手册

五． 实验步骤

本实验附件一节中展示了一段 Python 代码，实现了一个 UDP 服务器，该服务器还会模拟丢失 30% 的客户端数据包。请参考该代码，基于 Python 按照实验任务要求完成 Ping 程序的客户端。

注意：在运行客户端程序前，需要先运行服务器端代码。

编写成功后，使用客服端 Ping 程序经 UDP 向目标服务器发送 10 个 ping 报文。要求：

1. 使用 UDP 发送 ping 消息（注意：因为 UDP 是无连接协议，不需要建立连接。）；
2. 如果服务器在 1 秒内响应，则打印该响应消息；计算并打印每个数据包的往返时间 RTT（以秒为单位）；
3. 否则，打印“ 请求超时”（中英文皆可）。

在开发过程中，可以将客户端程序和服务器程序放在同一台电脑上进行测试。在完成代码调试后，可以尝试将客户端和服务器代码运行在不同网络环境，记录并分析结果。

六． 进阶任务

尝试修改代码，在程序运行结束时，计算所有 ping 消息的最小、最大和平均 RTT，并计算丢包率（丢失数据包在总数据包中所占有的百分比）。即构造出一个符合标准 Windows 版 Ping 程序工作模式的基于 UDP 版 Ping 程序。

七． 考核方法

本实验需提交一份实验报告和编写的代码文件。报告内容应当包括以下三个部分：

- 代码的说明；
- 不同环境下代码运行的结果；
- 对结果的分析和总结体会。

本实验评分标准：

1. 规定时间内完成实验报告 20 分；
2. 代码正确运行，20 分（不能正常运行 0 分）；
3. 实验报告格式整洁，20 分；
4. 实验报告中详细记录了实验过程，在实验中所遇到的问题以及解决方法，20 分；
5. 实验报告中仔细分析了实验结果，并能提出自己的改进措施，20 分。

八． 附件

基于 Python 的 UDP 套接字示例程序：

```
from socket import *
# 创建一个 UDP 套接字（SOCK_DGRAM）
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', 12000))
while True:
    rand = random.randint(0, 10)
    message, address = serverSocket.recvfrom(1024)
    message = message.upper()
    # 模拟 30% 的数据包丢失。
    if rand < 4:
        Continue
    serverSocket.sendto(message, address)
```

实验 1.2 TCP 通信与 Web 服务器

一． 实验目的

熟悉基于 Python 进行 TCP 套接字编程的基础知识，理解 HTTP 报文格式，能基于 Python 编写一个可以一次响应一个 HTTP 请求，并返回静态文件的简单 Web 服务器。

二． 实验内容

利用 Python 开发一个可以一次处理一个 HTTP 请求的 Web 服务器，该服务器可以接受并解析 HTTP 请求，然后从服务器的文件系统中读取被 HTTP 请求的文件，并根据该文件是否存在而向客户端发送正确的响应消息，

三． 实验原理、方法和手段

基于 TCP 的面向客户端/服务器在 Python 实现中的的工作流程是：

1. 首先在服务器端通过调用 `socket()` 创建套接字来启动一个服务器；
2. 服务器调用 `bind()` 绑定指定服务器的套接字地址（IP 地址 + 端口号）；
3. 服务器调用 `listen()` 做好侦听准备，同时规定好请求队列的长度；
4. 服务器进入阻塞状态，等待客户的连接请求；
5. 服务器通过 `accept()` 来接收连接请求，并获得客户的 socket 地址。
6. 在客户端通过调用 `socket()` 创建套接字；
7. 客户端调用 `connect()` 和服务器建立连接。
8. 连接建立成功后，客户端和服务器之间通过调用 `read()` 和 `write()` 来接收和发送数据。
9. 数据传输结束后，服务器和客户各自通过调用 `close()` 关闭套接字。

注意在不同的计算机语言实现中，上述调用的名字和具体工作流程可能略有不同。基于 Python 的 TCP 客户端/服务器具体工作流程如图1.2–1所示。

四． 实验条件

- 装有 python 环境的电脑两台；
- 局域网环境；
- 部分代码（附件中已给出）；

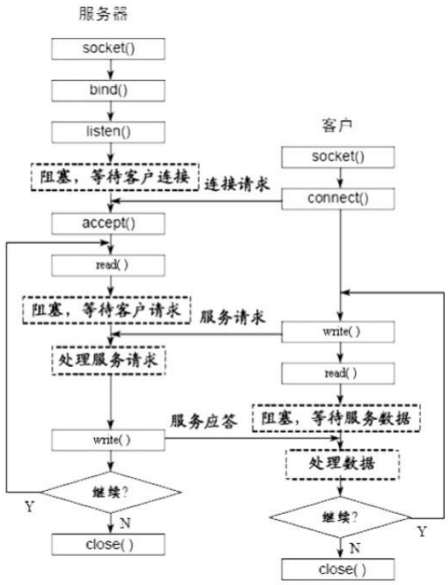

图 1.2–1 面向连接客户/服务器流程图

- Python 语言参考手册 – TCP 部分[1]；
- HTTP 协议参考手册[2]。

五． 实验步骤

本实验附件一节中展示了一份不完整的 Web 服务器框架代码，学生需要逐步填充代码中不完善的部分，并完成一个具有以下功能的简单 Web 服务器：

1. 服务器收到请求时能创建一个 TCP 套接字；
2. 可以通过这个 TCP 套接字接收 HTTP 请求；
3. 解析 HTTP 请求并在操作系统中确定客户端所请求的特定文件；
4. 从服务器的文件系统读取客户端请求的文件；
5. 当被请求文件存在时，创建一个由被请求的文件组成的“ 请求成功” HTTP 响应报文；
6. 当被请求文件不存在时，创建“ 请求目标不存在” HTTP 响应报文；
7. 通过 TCP 连接将响应报文发回客户端；

在开发过程中，可以使用浏览器访问运行在同一台电脑上的服务器程序进行测试。在完成代码调试后，可以尝试在不同主机上的使用浏览器发送请求测试服务器，分析并记录结果。

[1]可以参考 Python3 官方手册的 套接字部分，也可以查询其他相关手册

[2]可以参考 Mozilla 提供的 HTTP 概述，或者 HTTP 1.1 版的RFC 文档，也可以查询其他相关手册

六． 进阶任务

本实验中的 Web 服务器一次只能处理一个 HTTP 请求，请自行查阅线程知识，修改代码，实现一个能够同时处理多个请求的多线程服务器。

七． 注意事项及有关说明

客户端发来的 HTTP 请求中，URL 都是相对根“ /”的相对路径，因此需要将服务器文件系统中的某个地址映射为这个“ 根” 。为了简化工作，建议将服务器程序存放的路径作为这个根，这样运行服务器程序时，程序会自动从当前运行的路径开始查询文件。

例如：假设将“ HelloWorld.html”这个 html 文件放置在服务器程序文件存放目录中，服务器运行主机的 IP 地址为“ 123.234.12.34”，6789 为服务器监听的端口号。

则从 URL：http;//123.234.12.34:6789/HelloWorld.html 出发可以获取到“ HelloWorld.html”这个文件。

八． 考核方法

本实验需提交一份实验报告和编写的代码文件。报告内容应当包括以下三个部分：

- 代码的说明；
- 不同环境下代码运行的结果；
- 对结果的分析和总结体会。

本实验评分标准：

1. 规定时间内完成实验报告 20 分；
2. 代码正确运行，20 分（不能正常运行 0 分）；
3. 实验报告格式整洁，20 分；
4. 实验报告中详细记录了实验过程，在实验中所遇到的问题以及解决方法，20 分；
5. 实验报告中仔细分析了实验结果，并能提出自己的改进措施，20 分。

九． 附件

基于 Python 的 Web 服务器框架程序：

```
#import socket module
from socket import *
```

```
# 准备服务器端 socket（按需补充）
serverSocket = socket(AF_INET, SOCK_STREAM)
while True:
# 建立连接
    print 'Ready to serve...'
    connectionSocket, addr = # 按需补充
    try:
      message = # 按需补充
      filename = message.split()[1]
      f = open(filename[1:])
      # 通过 socket 发送 HTTP 头部
      outputdata = # 将请求文件的内容发送到客户端（按需补充）
      for i in range(0, len(outputdata)):
        connectionSocket.send(outputdata[i])
      connectionSocket.close()
    except IOError:
      # 发送未找到文件的响应消息（按需补充代码）
      # 关闭客户端 socket（按需补充代码）
serverSocket.close()
```