

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-停等协议、GBN协议和SR协议的设计与实现 | | | | | |
| 姓名 | 许坤彬 | | 院系 | 计算学部 | | |
| 班级 | 2203103 | | 学号 | 2022113586 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | G001 | | 实验时间 | 2024/10/17 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 1. 理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。  2. 理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于UDP 设计并实现一个 GBN 协议的过程与技术。 |
| 实验内容： |
| 1. 停等协议  （1）基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服  务器到客户的数据传输）。  （2）模拟引入数据包的丢失，验证所设计协议的有效性。  （3）改进所设计的停等协议，支持双向数据传输；  （4）基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。  2. GBN/SR协议  （1）基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。  （2）模拟引入数据包的丢失，验证所设计协议的有效性。  （3）改进所设计的 GBN 协议，支持双向数据传输；  （4）将所设计的 GBN 协议改进为 SR 协议。 |
| 实验过程： |
| 1. 停等协议的设计与实现  1.1 停等协议的数据分组格式与确认分组格式  停等协议的数据分组主要需要包含两部分的内容，一方面是当前数据包的序列号，另一方面是数据包的内容。数据包的内容采用utf-8的格式进行编码。  packet = f"{state}-{data\_packets[0]}".encode('utf-8')  确认分组的格式同样由两个部分构成，一方面是  1.1 基于UDP的简单停等协议实现  UDP的停等协议主要分为server端和client端两部分，数据报格式主要设计为“state-data\_packet”，这里的state是用来作为序列区分的序列号，停等协议的实现可以用状态机来进行描述。    发送的同时会为发送的每一个数据包设定定时器，当超时重新发送当前状态的数据包。发送之后进入等待ACK的状态，当收到ACK且ACK状态为1时，发送下一个数据包并且反转状态为0，否则重发数据包。  import socket  import time  import random  SERVER\_IP = '127.0.0.1'  SERVER\_PORT = 12345  CLIENT\_IP = "127.0.0.1"  CLIENT\_PORT = 12346  BUFFER\_SIZE = 1024  TIMEOUT = 2  # 超时时间（秒）  # 模拟丢包函数  def simulate\_packet\_loss():      return random.random() < 0.2  # 20%的概率丢包  def udp\_server():      server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      server\_socket.bind((SERVER\_IP, SERVER\_PORT))      print(f"服务器启动，等待客户端连接...")      # 数据包的内容      data\_packets = ["数据包1:哈", "数据包2：尔", "数据包3：滨", "数据包4：工", "数据包5：业",  "数据包6：大", "数据包7：学"]      # current\_packet = 0  # 当前发送的包序号      state = 0           # 初始序列号为0      while len(data\_packets) > 0 :          # 发送数据包格式：状态-          packet = f"{state}-{data\_packets[0]}".encode('utf-8')            # 模拟丢包          if simulate\_packet\_loss():              print(f"模拟丢失：{data\_packets[0]}")              continue          server\_socket.sendto(packet, (CLIENT\_IP, CLIENT\_PORT))          print(f"发送数据包：{data\_packets[0]}")          # 设置超时接收          server\_socket.settimeout(TIMEOUT)          try:              ack, client\_address = server\_socket.recvfrom(BUFFER\_SIZE)              ack = ack.decode('utf-8')                if ack == f"ACK{state}":                  print(f"收到 ACK：{ack}")                  data\_packets.pop(0) # 弹出待发送队列的列首                  if state == 0 :                      state = 1       # 收到ACK反转状态                  elif state == 1:                      state = 0              else:                  print(f"收到错误的 ACK：{ack}，重发当前数据包。")            except socket.timeout:              print(f"超时未收到 ACK{state}，重发数据包。")  if \_\_name\_\_ == "\_\_main\_\_":      udp\_server()  客户端的实现主要需要注意的是判断所接受的数据包序列号是否与自己想要的一致，如果一致则发送正确的ACK，否则不做任何处理，也不接受数据包，等待服务端超时后重新传输。  def udp\_client():      client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      client\_socket.bind((CLIENT\_IP, CLIENT\_PORT))      print("客户端启动，等待数据接收...")      expected\_packet = 0      try:          while True:              data, server\_address = client\_socket.recvfrom(BUFFER\_SIZE)              data = data.decode('utf-8')              sequence\_number, packet\_content = data.split('-')              if int(sequence\_number) == expected\_packet:                  print(f"收到正确的包：{packet\_content}")                  ack = f"ACK{expected\_packet}".encode('utf-8')                  client\_socket.sendto(ack, server\_address)                  if expected\_packet == 1:                      expected\_packet = 0                  elif expected\_packet == 0:                      expected\_packet = 1              else:                  print(f"接收到错误包，期望包号：{expected\_packet}")        except KeyboardInterrupt:          print("\n客户端已被中断，正在关闭...")      finally:          client\_socket.close()          print("客户端已关闭。")  1.2 丢包的模拟  使用随机数函数进行丢包的模拟，定义一个0~1之间的随机数，当随机数<0.2时只在控制台输出丢包，而不发送数据包。  # 模拟丢包函数  def simulate\_packet\_loss():      return random.random() < 0.2  # 20%的概率丢包  1.3 双向数据传输的实现  双向数据传输的实现需要将客户端代码和服务端代码整合到一起，并将运行服务端或者客户端所需要的IP、端口或者需要传输的数据包等参数改为函数接口。从而形成一个完整的停等协议。  import socket  import time  import random  BUFFER\_SIZE = 1024  TIMEOUT = 2  # 超时时间（秒）  # 模拟丢包函数  def simulate\_packet\_loss():      return random.random() < 0.2  # 20%的概率丢包  def udp\_server(sever\_ip, sever\_port, client\_ip, client\_port, data\_packets):      server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      server\_socket.bind((sever\_ip, sever\_port))      print(f"服务器启动，等待客户端连接...")      # 数据包的内容      # data\_packets = ["数据包1:小", "数据包2：狗", "数据包3：汪", "数据包4：汪", "数据包5：队"]      # current\_packet = 0  # 当前发送的包序号      state = 0           # 初始序列号为0      while len(data\_packets) > 0 :          # 发送数据包格式：状态-数据          packet = f"{state}-{data\_packets[0]}".encode('utf-8')            # 模拟丢包          if simulate\_packet\_loss():              print(f"模拟丢失：{data\_packets[0]}")              continue          server\_socket.sendto(packet, (client\_ip, client\_port))          print(f"发送数据包：{data\_packets[0]}")          # 设置超时接收          server\_socket.settimeout(TIMEOUT)          try:              ack, client\_address = server\_socket.recvfrom(BUFFER\_SIZE)              ack = ack.decode('utf-8')                if ack == f"ACK{state}":                  print(f"收到 ACK：{ack}")                  data\_packets.pop(0) # 弹出待发送队列的列首                  if state == 0 :                      state = 1       # 收到ACK反转状态                  elif state == 1:                      state = 0              else:                  print(f"收到错误的 ACK：{ack}，重发当前数据包。")            except socket.timeout:              print(f"超时未收到 ACK{state}，重发数据包。")      def udp\_client(client\_ip, client\_port):      client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      client\_socket.bind((client\_ip, client\_port))      print("客户端启动，等待数据接收...")      expected\_packet = 0      try:          while True:              data, server\_address = client\_socket.recvfrom(BUFFER\_SIZE)              data = data.decode('utf-8')              sequence\_number, packet\_content = data.split('-')              if int(sequence\_number) == expected\_packet:                  print(f"收到正确的包：{packet\_content}")                  ack = f"ACK{expected\_packet}".encode('utf-8')                  client\_socket.sendto(ack, server\_address)                  if expected\_packet == 1:                      expected\_packet = 0                  elif expected\_packet == 0:                      expected\_packet = 1              else:                  print(f"接收到错误包，期望包号：{expected\_packet}")        except KeyboardInterrupt:          print("\n客户端已被中断，正在关闭...")      finally:          client\_socket.close()          print("客户端已关闭。")  在使用上，通过线程调用的形式完成服务端和客户端的调用，进而实现服务端和客户端之间的双线信息传递。在这里为用户端开放两个线程，一个用于发送信息，另一个用于接收信息。同样为客户端开放两个线程，分别用于发送和接收信息。  import wait as wait  import threading  SERVER\_IP = '127.0.0.1'  SERVER\_SEND\_PORT = 12345  SERVER\_RECV\_PORT = 12347  CLIENT\_IP = "127.0.0.1"  CLIENT\_SEND\_PORT = 12346  CLIENT\_RECV\_PORT = 12348  # 模拟要传输的数据  data\_packet1 = ["数据包1:哈", "数据包2：尔", "数据包3：滨", "数据包4：工", "数据包5：业",  "数据包6：大", "数据包7：学"]  data\_packet2 = ["数据包1:规", "数据包2：格", "数据包3：严", "数据包4：格", "数据包5：功",  "数据包6：夫", "数据包7：到", "数据包8：家"]  Thread\_Server\_send = threading.Thread(target=wait.udp\_server, args=(SERVER\_IP, SERVER\_SEND\_PORT, CLIENT\_IP, CLIENT\_RECV\_PORT, data\_packet1, ))  Thread\_Server\_recv = threading.Thread(target=wait.udp\_client, args=(SERVER\_IP, SERVER\_RECV\_PORT, ))  Thread\_Client\_send = threading.Thread(target=wait.udp\_server, args=(CLIENT\_IP, CLIENT\_SEND\_PORT, SERVER\_IP, SERVER\_RECV\_PORT, data\_packet2, ))  Thread\_Client\_recv = threading.Thread(target=wait.udp\_client, args=(CLIENT\_IP, CLIENT\_RECV\_PORT, ))  Thread\_Server\_send.start()  Thread\_Client\_recv.start()  Thread\_Client\_send.start()  Thread\_Server\_recv.start()  1.4 C/S应用的实现  C/S文件传输应用的实现需要在停等协议的基础上，增加文件读写和分块的操作。在服务端，首先需要打开需要传递的文件，并获取文件的大小，进而计算文件传输需要多少数据包才能实现。接下来再分块读取文件内容，并获得文件分块列表。并按照列表的顺序发送每个数据包。同时，在读取到文件末尾时，需要加上一个EOF的标识，用来告知客户端已经完成了发送，让客户端停止接收。  在客户端则主要需要在接收到数据包的同时需要进行文件的写，当客户端拆分得到数据包之后，将序列号转换为整数进行判断，看是不是客户端想要的数据包，随后将接收到的数据块再写入到文件当中。之后再发送ACK，这样就可以实现基于停等协议的C/S文件传输应用。  # server  import socket  import os  import random  SERVER\_IP = '127.0.0.1'  SERVER\_PORT = 12345  CLIENT\_IP = '127.0.0.1'  CLIENT\_PORT = 12346  BUFFER\_SIZE = 1024  TIMEOUT = 2  # 超时时间（秒）  FILE\_PATH = 'server\_file.txt'  # 要发送的文件  # 模拟丢包函数  def simulate\_packet\_loss():      return random.random() < 0.2  # 20%的概率丢包  def udp\_server():      # 创建socket套接字，并且绑定在本地端口上      server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      server\_socket.bind((SERVER\_IP, SERVER\_PORT))      print(f"服务器启动，等待客户端连接...")      try:          # 首先打开需要传递的文件          with open(FILE\_PATH, 'rb') as file:              # 获取文件大小              file\_size = os.path.getsize(FILE\_PATH)              # 计算文件需要多少数据包              num\_packets = (file\_size // BUFFER\_SIZE) + 1              state = 0  # 初始序列号为0                # 根据文件读取结果确定什么时候结束循环              for i in range(num\_packets):                  # 读取文件的一块数据并且分别进行封装                  file\_chunk = file.read(BUFFER\_SIZE)                  packet = f"{state}-".encode('utf-8') + file\_chunk                  # # 模拟丢包                  # if simulate\_packet\_loss():                  #     print(f"模拟丢失：数据包{state}")                  #     continue                  # 向客户端发送对应的数据包                  server\_socket.sendto(packet, (CLIENT\_IP, CLIENT\_PORT))                  print(f"发送数据包{state}")                  # 设置超时接收                  server\_socket.settimeout(TIMEOUT)                  try:                      ack, client\_address = server\_socket.recvfrom(BUFFER\_SIZE)                      ack = ack.decode('utf-8')                      if ack == f"ACK{state}":                          print(f"收到 ACK：{ack}")                          state = 1 - state  # 切换状态                      else:                          print(f"收到错误的 ACK：{ack}，重发当前数据包。")                  except socket.timeout:                      print(f"超时未收到 ACK{state}，重发数据包。")              # 文件传输完成后，单独发送结束标记              end\_packet = "EOF".encode('utf-8')              server\_socket.sendto(end\_packet, (CLIENT\_IP, CLIENT\_PORT))              print("文件传输结束标记已发送。")      except FileNotFoundError:          print("要发送的文件不存在。")      finally:          server\_socket.close()  if \_\_name\_\_ == "\_\_main\_\_":      udp\_server()  # client  import socket  CLIENT\_IP = '127.0.0.1'  CLIENT\_PORT = 12346  BUFFER\_SIZE = 1024  FILE\_PATH = 'received\_file.txt'  # 接收后保存的文件  def udp\_client():      # 创建客户端套接字      client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      client\_socket.bind((CLIENT\_IP, CLIENT\_PORT))      print("客户端启动，等待文件接收...")      expected\_packet = 0      try:          with open(FILE\_PATH, 'wb') as file:              while True:                  data, server\_address = client\_socket.recvfrom(BUFFER\_SIZE + 10)  # 增加缓冲区以处理数据包头                    # 当内容为结束符号时                  if data.decode('utf-8') == "EOF":                      print("文件接收完毕，收到结束标记。")                      break  # 停止接收                  # 使用'-'分隔符将序列号和文件块分开                  sequence\_number\_str, file\_chunk = data.decode('utf-8').split('-', 1)                  try:                      sequence\_number = int(sequence\_number\_str)  # 将序列号转换为整数                  except ValueError:                      print(f"解析序列号失败，收到无效数据：{sequence\_number\_str}")                      continue                  if sequence\_number == expected\_packet:                      # 写入接收到的文件块                      file.write(file\_chunk.encode('utf-8'))  # 将文件块写入文件                      print(f"收到数据包：{expected\_packet}")                      # 发送 ACK 确认                      ack = f"ACK{expected\_packet}".encode('utf-8')                      client\_socket.sendto(ack, server\_address)                      expected\_packet = 1 - expected\_packet  # 切换期望的数据包序号                  else:                      print(f"接收到错误包，期望包号：{expected\_packet}")        except KeyboardInterrupt:          print("\n客户端已被中断，正在关闭...")      finally:          client\_socket.close()          print("客户端已关闭。")  if \_\_name\_\_ == "\_\_main\_\_":      udp\_client()  2. |
| 实验结果： |
| 采用演示截图、文字说明等方式，给出本次实验的实验结果。 |
| 问题讨论： |
|  |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。 |