实验报告

PART A

本实验旨在实现一个简单的网络编程示例，包括客户端和服务端的代码实现。我选用的语言是C语言与Python。

C语言实现

客户端

实现逻辑

首先利用getaddrinfo()函数获取服务器的地址信息，之后使用循环遍历获取到的地址信息，尝试连接服务器。在循环中，首先创建套接字（socket()），然后使用connect()函数连接到服务器。连接完服务器后释放地址信息（freeaddrinfo()），接下来，通过循环从标准输入读取数据，并使用send()函数将数据发送给服务器。最后，待发送完信息后关闭套接字（close()）。

代码实现

```
nt client(char *server_ip, char *server_port)
{
  int sockfd, ret;
  struct addrinfo hints, *servinfo, *p;
  char send_buffer[SEND_BUFFER_SIZE];
  size_t bytes_read;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;

  if ((ret = getaddrinfo(server_ip, server_port, &hints, &servinfo)) != 0)
  {
    fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(ret));
    return 1;
  }

  // 遍历所有结果，并尝试连接
  for (p = servinfo; p != NULL; p = p->ai_next)
  {
    if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1)
    {
      perror("client: socket");
      continue;
    }

    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1)
    {
```

```
      close(sockfd);
      perror("client: connect");
      continue;
    }

    break; // 连接成功，退出循环
  }

  if (p == NULL)
  {
    fprintf(stderr, "client: failed to connect\n");
    return 2;
  }

  freeaddrinfo(servinfo); // 释放地址信息

  // 从标准输入读取数据，并发送给服务器
  while ((bytes_read = fread(send_buffer, 1, sizeof(send_buffer), stdin)) > 0)
  {
    if (send(sockfd, send_buffer, bytes_read, 0) == -1)
    {
      perror("send");
      break;
    }
  }

  close(sockfd); // 关闭套接字

  return 0;
}
```

服务端

实现逻辑

首先，我使用getaddrinfo()函数获取本机的地址信息，其中包括服务器的IP地址（设置为NULL，表示使用本机IP）和端口号。然后，使用循环遍历获取到的地址信息，尝试绑定套接字。在循环中，首先创建套接字（socket()），然后使用bind()函数将套接字与地址绑定，使用listen()函数监听套接字，设置连接请求队列的长度。之后，进入无限循环，接受客户端连接请求（accept()）。当有新的连接请求到达时，accept()会返回一个新的套接字描述符（new_fd），用于与客户端进行通信，并使用recv()函数接收从客户端发送的数据，并将接收到的数据写入标准输出（stdout）。每次请求结束，关闭套接字（new_fd），最后，关闭监听套接字（sockfd）。

代码实现

```
int server(char *server_port)
{
  int sockfd, new_fd;
  struct addrinfo hints, *servinfo, *p;
  struct sockaddr_storage client_addr;
  socklen_t addr_size;
  int ret;
  char recv_buffer[RECV_BUFFER_SIZE];

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
```

```
hints.ai_flags = AI_PASSIVE;

if ((ret = getaddrinfo(NULL, server_port, &hints, &servinfo)) != 0)
{
  fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(ret));
  return 1;
}

// 遍历所有结果，并尝试绑定套接字
for (p = servinfo; p != NULL; p = p->ai_next)
{
  if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1)
  {
    perror("server: socket");
    continue;
  }

  if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1)
  {
    close(sockfd);
    perror("server: bind");
    continue;
  }

  break; // 绑定成功，退出循环
}

if (p == NULL)
{
  fprintf(stderr, "server: failed to bind\n");
  return 2;
}

freeaddrinfo(servinfo); // 释放地址信息

if (listen(sockfd, QUEUE_LENGTH) == -1)
{
  perror("listen");
  return 3;
}

// 注册 SIGINT 信号处理函数
struct sigaction sa;
sa.sa_handler = sigint_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
if (sigaction(SIGINT, &sa, NULL) == -1)
{
  perror("sigaction");
  return 4;
}

while (1)
{
  addr_size = sizeof client_addr;
  new_fd = accept(sockfd, (struct sockaddr *)&client_addr, &addr_size);
  if (new_fd == -1)
  {
```

```
      perror("accept");
      continue;
    }

    // 从客户端接收消息，并打印到标准输出
    while (1)
    {
      ret = recv(new_fd, recv_buffer, sizeof(recv_buffer), 0);
      if (ret == -1)
      {
        perror("recv");
        break;
      }
      else if (ret == 0)
      {
        break;
      }

      fwrite(recv_buffer, 1, ret, stdout);
      fflush(stdout);
    }

    close(new_fd); // 关闭新的套接字
  }

  close(sockfd); // 关闭监听套接字

  return 0;
}
```

Python

客户端

实现逻辑

- 创建一个套接字（sockfd）并与服务器建立连接。
- 通过循环，从标准输入读取数据（sys.stdin.buffer.read(SEND_BUFFER_SIZE)），并将其发送给服务器（sockfd.sendall(send_data)）。
- 关闭套接字。

代码实现

```
def client(server_ip, server_port):
    try:
        # 创建套接字并连接服务器
        sockfd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sockfd.connect((server_ip, server_port))

        # 从标准输入读取数据，并发送给服务器
        while True:
            send_data = sys.stdin.buffer.read(SEND_BUFFER_SIZE)
            if not send_data:
```

```
                break
            sockfd.sendall(send_data)

        # 关闭套接字
        sockfd.close()
    except socket.error as e:
        print("Socket error: {}".format(e))
        sys.exit(1)
```

服务端

实现逻辑

- 创建一个套接字（sockfd）并绑定到指定端口。
- 监听套接字，等待客户端连接。
- 接受客户端连接后，创建一个新的套接字（new_fd）和客户端进行通信。
- 通过循环，从新的套接字接收数据（new_fd.recv(RECV_BUFFER_SIZE)），并将其打印到标准输出（sys.stdout.buffer.write(recv_data)）。
- 关闭新的套接字。

代码实现

```
def server(server_port):
    try:
        # 创建套接字并绑定到指定端口
        sockfd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sockfd.bind(("", server_port))

        # 监听套接字
        sockfd.listen(QUEUE_LENGTH)

        while True:
            # 接受客户端连接
            new_fd, addr = sockfd.accept()

            # 从客户端接收消息，并打印到标准输出
            while True:
                recv_data = new_fd.recv(RECV_BUFFER_SIZE)
                if not recv_data:
                    break
                sys.stdout.buffer.write(recv_data)
                sys.stdout.flush()

            # 关闭新的套接字
            new_fd.close()
    except socket.error as e:
        print("Socket error: {}".format(e))
        sys.exit(1)
```

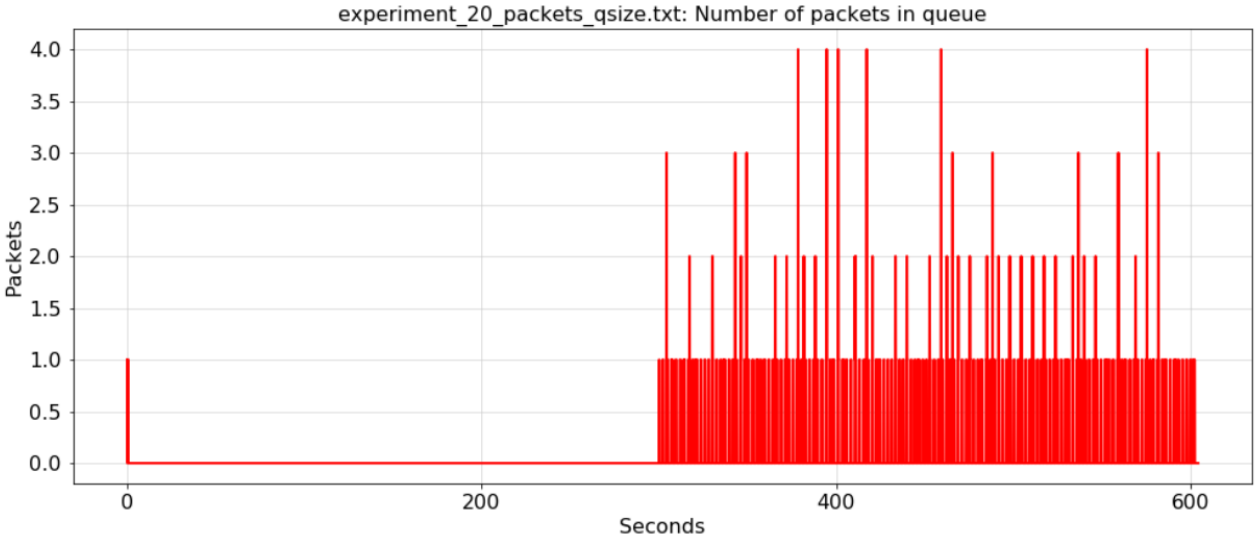
实验结果

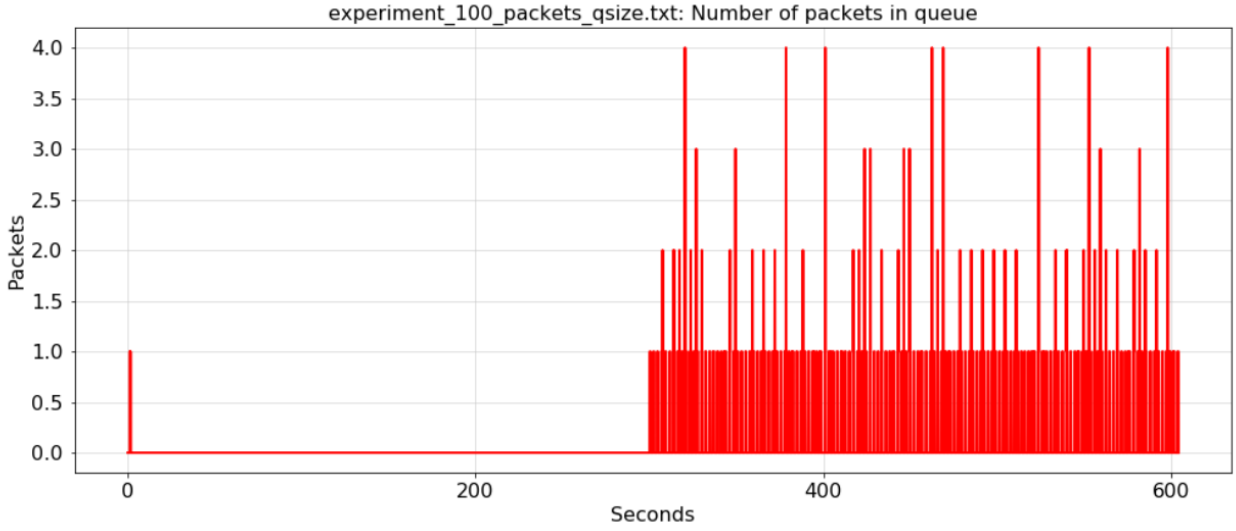
```
19. TEST SERVER INFINITE LOOP (multiple sequential clients to same server)

SUCCESS: Message received matches message sent!

20. TEST SERVER QUEUE (overlapping clients to same server)

SUCCESS: Message received matches message sent!

================================================================

TESTS PASSED: 20/20
vagrant@netruc:/vagrant/client_server$
```

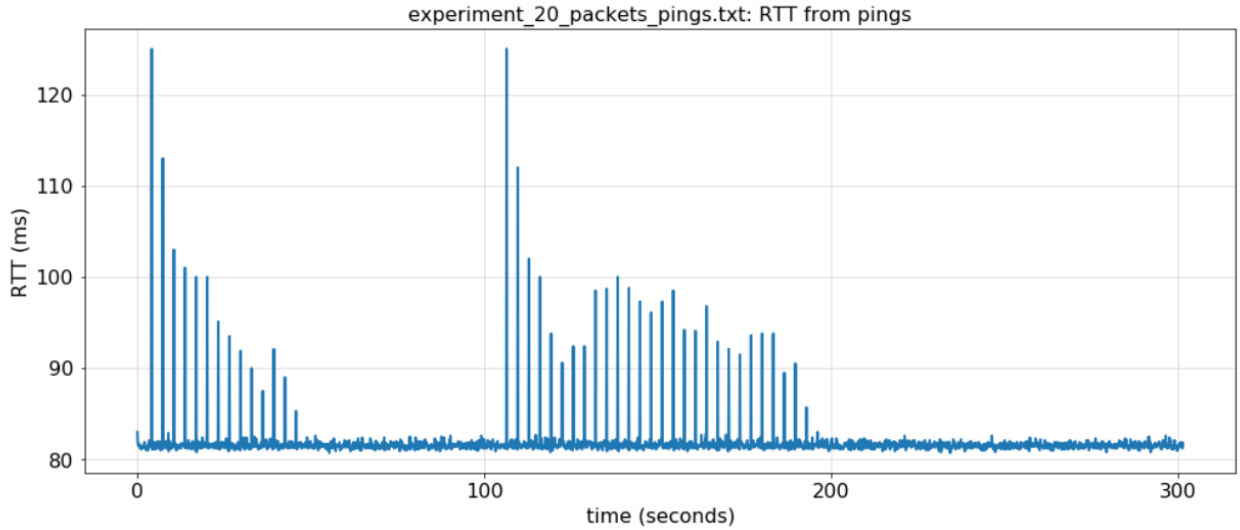
PART B

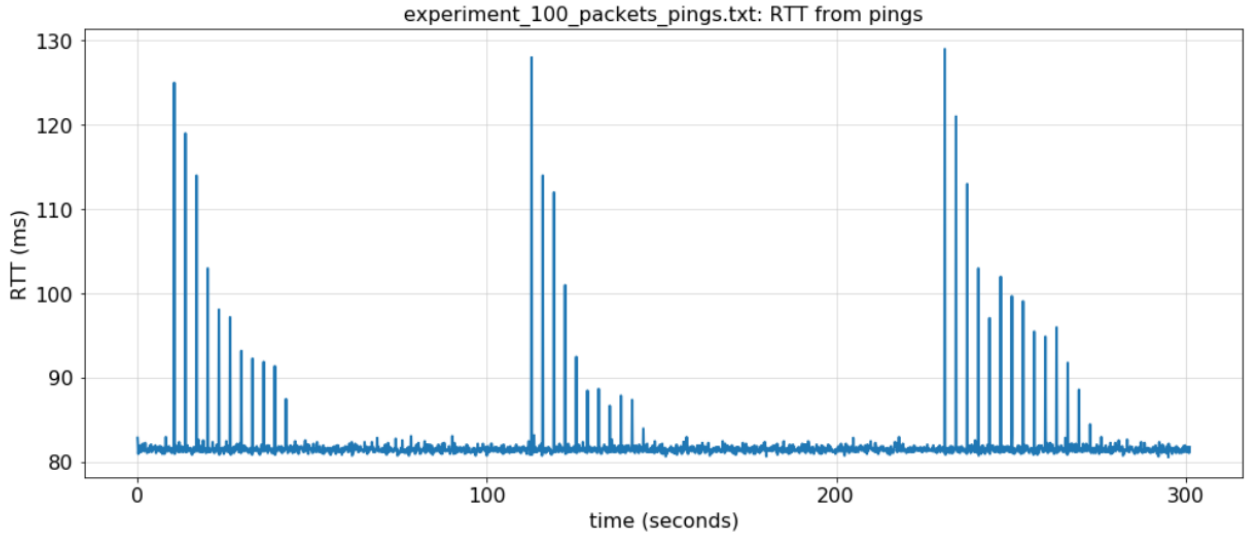
在本实验中，你将创建自己的模拟网络，以动态地研究TCP，以及网络操作员看似微不足道的配置决策是如何对网络性能产生重大影响的。但由于虚拟机配置原因，无法测试cwnd大小。

实验结果






experiment_20_packets_pings.txt: RTT from pings
RTT (ms)
120
110
100
90
80
0
100
200
300
time (seconds)


experiment_100_packets_pings.txt: RTT from pings
RTT (ms)
130
120
110
100
90
80
0
100
200
300
time (seconds)