

软件学院

计算机网络课程设计

内网穿透的实现及基于内网穿透的服务搭建

|  |  |
| --- | --- |
| 姓 名： |  |
| 学 号： |  |
| 班 级： |  |
| 邮 箱： |  |
| 完成日期： |  |

**目 录**

[1 选题介绍和需求分析 1](#_Toc166748172)

[1.1 为什么要实现内网穿透——内网穿透能做什么 1](#_Toc166748173)

[1.2 内网穿透的需求分析 1](#_Toc166748174)

[2 实现方式分析[1] 2](#_Toc166748175)

[2.1 服务端实现方式 2](#_Toc166748176)

[2.2 客户端实现方式 2](#_Toc166748177)

[3 代码展示 3](#_Toc166748178)

[3.1 代码结构 3](#_Toc166748179)

[3.2 TCP转发代码分析 3](#_Toc166748180)

[3.3 客户端重点代码分析 4](#_Toc166748181)

[3.4 服务端重点代码分析 5](#_Toc166748182)

[4 结果展示 6](#_Toc166748183)

[4.1 基于内网穿透的文件传输服务 6](#_Toc166748184)

[4.2 基于内网穿透的网站服务 7](#_Toc166748185)

[5 总结 7](#_Toc166748186)

[6 参考文献 8](#_Toc166748187)

[7 源代码 8](#_Toc166748188)

[7.1 frp-client.py 8](#_Toc166748189)

[7.2 frp-server.py 10](#_Toc166748190)

# 选题介绍和需求分析

## 为什么要实现内网穿透——内网穿透能做什么

由于IPv4地址的稀缺性，目前我们对互联网的访问经过了层层NAT，即IPv4公网上的设备不能直接通过IP地址访问我们接入互联网的设备。当我们访问互联网时，NAT设备会对IP数据报进行转发，将内网的IP和端口与公网的IP和端口相对应。例如，电脑连接路由器的WiFi，电脑的IP地址为192.168.1.100，而真正访问公网的IP地址为115.154.17.24。如此一来，如果我们要在电脑上搭建网站，外界并不能通过192.168.1.100来访问（因为这是局域网地址），也不能通过115.154.17.24来访问（因为这是路由器或NAT设备的地址），此时我们就需要内网穿透服务。

FRP（Fast Reverse Proxy）即快速反向代理服务，可以将内网服务以安全、便捷的方式通过具有公网 IP 节点的中转暴露到公网。其应用场景有：

在局域网中的主机上提供网页、文件等服务；

从互联网的其他位置访问和控制（远程连接）局域网中的计算机、监控、存储设备等；

与其他局域网中的主机通信，包括游戏联机、文件传输等。

## 内网穿透的需求分析

综上所述，内网穿透的主要需求是将局域网中的主机上搭建的服务提供到公网上去。因此，在进行内网穿透前我们必须具有一个公网IP地址（可能是服务器，也可能是其他主机），从而将服务映射到那个公网IP地址上去。在公网设备上，FRP服务监听某个端口的TCP请求，并将其转发给局域网设备（这要求局域网设备上的FRP服务发起一个持续的TCP连接），并将局域网设备的回复转发出去。这就是FRP服务所需的功能。

还有一种内网穿透能够实现无其他设备参与的两个局域网主机之间的通信。它将进行TCP/UDP打洞，即公网设备上的服务将让主机相互知道对方暴露在公网上的IP和端口，从而使它们建立连接。但由于互联网上NAT设备的复杂性，有些NAT可能会丢掉数据包从而造成一定的失败率。

在此次大作业中，我主要实现第一种内网穿透。

# 实现方式分析[1]

我们可以采取Python和Socket编程来控制TCP连接和TCP包的转发。FRP应当针对服务端（有公网IP）和客户端（局域网中）两种分别实现，提供不同的服务。

## 服务端实现方式

在服务端，FRP进行以下服务：

服务一经开启，监听与客户端通信的接口；

如果该端口有TCP请求，尝试与客户端建立一个长TCP连接。

监听用户端口；

如果该端口有TCP请求，通过与客户端之间的TCP长连接向客户端发送一条控制请求；并与客户端新建一条工作TCP连接，并将其与用户的TCP请求绑定（即转发数据）。

## 客户端实现方式

在客户端，FRP进行以下服务：

服务一经开启，先向服务端进行TCP请求以建立TCP长连接。

如果收到服务端发来的TCP控制请求，则与客户端上的用户服务建立一个TCP连接，再与服务端建立一个工作TCP连接，并将它们绑定（即转发数据）。

# 代码展示

## 代码结构

FRP服务分为客户端和服务端，因此由两个文件frp-client.py和frp-server.py构成。两个文件均需要使用的功能有：tcp\_transfer即TCP段的转发、join即创建线程进行双向TCP转发。

## TCP转发代码分析

**def** tcp\_transfer(conn\_receiver, conn\_sender):

**while** **True**:

**try**:

data = conn\_receiver.recv(10240)

**except** Exception **as** e:

print(**'Failed to receive data:'**, e)

**break**

**if** **not** data:

print(**'No data received.'**)

**break**

**try**:

conn\_sender.sendall(data)

**except** Exception:

print(**'Failed sending data.'**, e)

**break**

conn\_receiver.close()

conn\_sender.close()

**return**

**def** join(connA, connB):

threading.Thread(target=tcp\_transfer, args=(connA, connB)).start()

threading.Thread(target=tcp\_transfer, args=(connB, connA)).start()

**return**

tcp\_transfer函数的作用是在两个socket连接间进行数据交换。它尝试从一个连接接收数据，然后发送给另外一个连接，实现数据的转发。如果出现了异常或者没有数据要接收，那么就会关闭这两个连接，函数执行完毕。join函数的作用是创建两个线程并启动，这两个线程用于管理两个方向的数据流，确保了两个连接之间的双向数据传输。

## 客户端重点代码分析

**def** heartbeat(self):

print(**"启动tcp连接保持"**)

**while** **True**:

**if** self.server\_fd **is** **not** **None**:

self.server\_fd.send(struct.pack(**'i'**, 1))

time.sleep(9)

**def** maitainConPool(self):

print(**"启动tcp连接池"**)

pool\_size = 0

**while** **True**:

**if** len(self.workConnPool)<pool\_size:

print(1)

workConn = socket.create\_connection((self.serverhost,self.serverport))

targetConn = socket.create\_connection((self.targethost, self.targetport))

join(targetConn,workConn)

self.workConnPool.append(workConn)

**def** handle\_controller\_data(self,server\_fd, mask):

**try**:

data= server\_fd.recv(4)

**if** data:

cmd = struct.unpack(**'i'**,data)[0]

print(**'收到控制报文:'**,cmd)

**if** cmd ==2:

**if** len(self.workConnPool)>0:

workConn = self.workConnPool.pop()

**else**:

targetConn = socket.create\_connection((self.targethost, self.targetport))

workConn = socket.create\_connection((self.serverhost,self.serverport))

join(targetConn,workConn)

workConn.sendall(struct.pack(**'i'**,2))

print(**"建立工作tcp"**)

**except** IOError:

**pass**

**def** run(self):

print(**'FRP已启动'**)

**while** **True**:

events = sel.select()

**for** key, mask **in** events:

callback = key.data

callback(key.fileobj, mask)

其中，sel是一个selectors.DefaultSelector对象，用于批量监控多个文件描述符的 I/O 变化（如网络套接字、管道等），当文件描述符状态改变时，程序立即收到通知。heartbeat(self) 函数确保客户端与服务器之间的持续连接。maitainConPool(self) 函数启动tcp连接池，该如果连接池中的连接数小于pool\_size，就创建workConn连接到服务器和targetConn连接到目标主机，然后调用join方法。handle\_controller\_data(self,server\_fd, mask) 函数：接收服务器端受控消息和处理情况，比如收到连接建立指令等。它每次循环接收服务器端发送的指令，然后根据指令进行相应的操作。run(self) 函数用于监听socket状态并分派处理函数，确保了在具有请求时即时响应。

## 服务端重点代码分析

*# 收到用户tcp*

**def** accept\_connection(self,sock, mask):

userConn, addr = self.sock.accept()

userConn.setblocking(**True**)

self.userConns.append(userConn)

print(**'收到用户请求'**)

**if** self.frp\_cmd\_conn **is** **None**:

print(**'不存在tcp控制连接'**)

**return**

**try**:

self.frp\_cmd\_conn.send(struct.pack(**'i'**,2)) *#建立新的tcp*

**except** IOError **as** err:

print(err)

**pass**

**def** accept\_frp\_connection(self,sock, mask):

frp\_conn, addr = sock.accept()

frp\_conn.setblocking(**False**)

sel.register(frp\_conn, selectors.EVENT\_READ, self.handle\_controller\_data)

**def** handle\_controller\_data(self,frp\_conn, mask):

**try**:

data = frp\_conn.recv(4)

**if** data:

cmd = struct.unpack(**'i'**,data)[0]

print(**"收到控制报文；"**,cmd)

**if** cmd ==2:

sel.unregister(frp\_conn)

userConn = self.userConns.pop()

frp\_conn.setblocking(**True**)

join(userConn,frp\_conn)

**elif** cmd ==1 **and** self.frp\_cmd\_conn!=frp\_conn:

self.frp\_cmd\_conn = frp\_conn

threading.Thread(target=self.heartbeat).start()

**except** IOError **as** err:

**pass**

与客户端基本一致的代码不再重述。accept\_connection(self,sock, mask) 函数，用于接收客户端的连接请求，并将客户端的连接对象保存到 userConns 列表，在有有效的 frp\_cmd\_conn 时，向其发送建立新连接的请求。accept\_frp\_connection(self,sock, mask) 函数，用于接受 frp 连接请求，并将新的 frp 连接注册到 selector 上等待读事件。handle\_controller\_data(self,frp\_conn, mask) 函数，根据收到的命令做出不同的处理，其中包括接收心跳数据和从 userConns 中取出连接进行数据转发。run(self) 函数，用于不断监控所有注册到 selector 上的文件描述符，并调用相应的回调函数处理对应的事件。

# 结果展示

将服务端代码拷贝至阿里云服务器上并运行，设定与客户端通信的端口为7889，对外提供服务的端口是7890，服务器公网ip是60.205.127.51。在本机上设定对应服务的地址（localhost:xxxx）和服务端地址（60.205.127.51:7889）并运行客户端代码。

## 基于内网穿透的文件传输服务

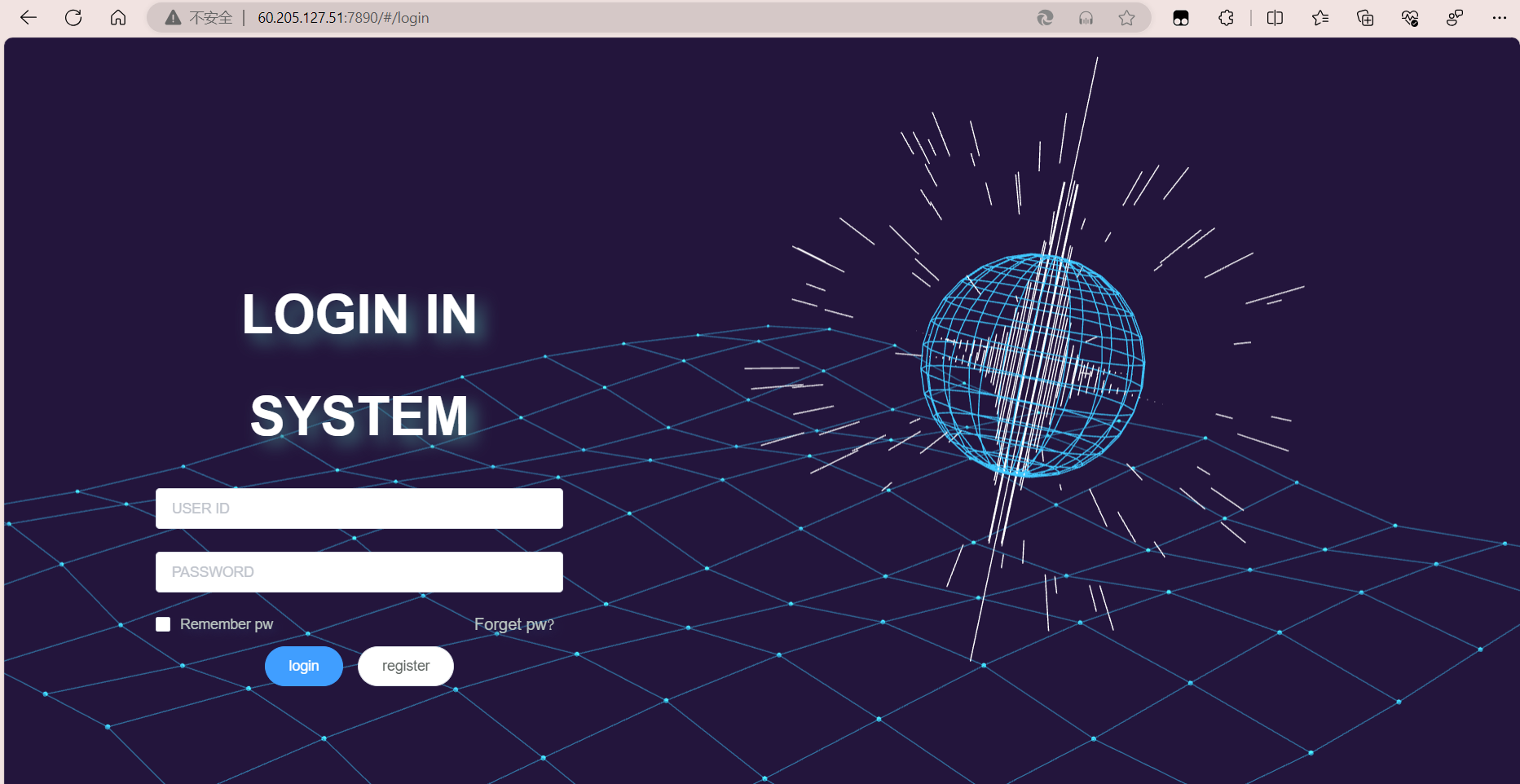
使用python指令python -m http.server可以将本地目录中的文件开放至网络端口上，如图所示。



该服务将本地文件共享至公网上供其他设备访问。

## 基于内网穿透的网站服务

利用FRP，可以将vue和springboot实现的Java Web项目共享至公网上。



# 总结

在本课程设计中，我实现了FRP内网穿透服务并将其应用于网页搭建、文件传输、游戏联机等场景。基于此课程设计，我对计算机网络的五层体系架构、Socket网络编程、TCP/IP协议等均有进一步的认知。同时，本课程设计的结果可以有效应用于使网络的使用场景中，是一个优秀的互联网小工具。

# 参考文献

[1] 内网穿透工具frp核心架构原理分析. [EB/OL] usualheart. 2022-02-20. https://blog.csdn.net/usualheart/article/details/123032372

[2] 基于frp原理实现的轻量级内网穿透工具frp-python [EB/OL] usualheart. 2022-02-23. https://blog.csdn.net/usualheart/article/details/123101274

# 源代码

## frp-client.py

**import** socket, time, threading, struct, selectors

**def** tcp\_transfer(conn\_receiver, conn\_sender):

**while** **True**:

**try**:

data = conn\_receiver.recv(10240)

**except** Exception **as** e:

print(**'Failed to receive data:'**, e)

**break**

**if** **not** data:

print(**'No data received.'**)

**break**

**try**:

conn\_sender.sendall(data)

**except** Exception:

print(**'Failed sending data.'**, e)

**break**

conn\_receiver.close()

conn\_sender.close()

**return**

**def** join(connA, connB):

threading.Thread(target=tcp\_transfer, args=(connA, connB)).start()

threading.Thread(target=tcp\_transfer, args=(connB, connA)).start()

**return**

sel = selectors.DefaultSelector()

**class** Frpclient():

**def** \_\_init\_\_(self, serverhost,serverport, targethost,targetport):

self.targethost=targethost

self.targetport=targetport

self.serverhost=serverhost

self.serverport=serverport

self.server\_fd = socket.create\_connection((self.serverhost,self.serverport))

self.server\_fd.sendall(struct.pack(**'i'**, 1))

self.server\_fd.setblocking(**False**)

sel.register(self.server\_fd,selectors.EVENT\_READ,self.handle\_controller\_data)

self.workConnPool = []

threading.Thread(target=self.maitainConPool).start()

threading.Thread(target=self.heartbeat).start()

**def** heartbeat(self):

print(**"启动tcp连接保持"**)

**while** **True**:

**if** self.server\_fd **is** **not** **None**:

self.server\_fd.send(struct.pack(**'i'**, 1))

time.sleep(9)

**def** maitainConPool(self):

print(**"启动tcp连接池"**)

pool\_size = 0

**while** **True**:

**if** len(self.workConnPool)<pool\_size:

print(1)

workConn = socket.create\_connection((self.serverhost,self.serverport))

targetConn = socket.create\_connection((self.targethost, self.targetport))

join(targetConn,workConn)

self.workConnPool.append(workConn)

**def** handle\_controller\_data(self,server\_fd, mask):

**try**:

data= server\_fd.recv(4)

**if** data:

cmd = struct.unpack(**'i'**,data)[0]

print(**'收到控制报文:'**,cmd)

**if** cmd ==2:

**if** len(self.workConnPool)>0:

workConn = self.workConnPool.pop()

**else**:

targetConn = socket.create\_connection((self.targethost, self.targetport))

workConn = socket.create\_connection((self.serverhost,self.serverport))

join(targetConn,workConn)

workConn.sendall(struct.pack(**'i'**,2))

print(**"建立工作tcp"**)

**except** IOError:

**pass**

**def** run(self):

print(**'FRP已启动'**)

**while** **True**:

events = sel.select()

**for** key, mask **in** events:

callback = key.data

callback(key.fileobj, mask)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

remothost = input(**"请输入服务端的ip地址:"**)

remoteport = int(input(**"请输入服务端的端口:"**))

targethost = input(**"请输入服务主机的ip地址:"**)

targetport = int(input(**"请输入服务主机的端口:"**))

print(**'正在启动FRP...'**)

frpclient = Frpclient(remothost,remoteport, targethost,targetport)

frpclient.run()

print(**'FRP已关闭'**)

## frp-server.py

**import** socket, time, threading, struct, selectors

**def** tcp\_transfer(conn\_receiver, conn\_sender):

**while** **True**:

**try**:

data = conn\_receiver.recv(10240)

**except** Exception **as** e:

print(**'Failed to receive data:'**, e)

**break**

**if** **not** data:

print(**'No data received.'**)

**break**

**try**:

conn\_sender.sendall(data)

**except** Exception:

print(**'Failed sending data.'**, e)

**break**

conn\_receiver.close()

conn\_sender.close()

**return**

**def** join(connA, connB):

threading.Thread(target=tcp\_transfer, args=(connA, connB)).start()

threading.Thread(target=tcp\_transfer, args=(connB, connA)).start()

**return**

sel = selectors.DefaultSelector()

**class** Frpserver(threading.Thread):

**def** \_\_init\_\_(self, port, targetport):

threading.Thread.\_\_init\_\_(self)

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.bind((**'0.0.0.0'**, port))

self.sock.setblocking(**False**)

self.sock.listen(100)

sel.register(self.sock,selectors.EVENT\_READ,self.accept\_connection)

frp\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

frp\_sock.bind((**'0.0.0.0'**, targetport))

frp\_sock.setblocking(**False**)

frp\_sock.listen(100)

sel.register(frp\_sock,selectors.EVENT\_READ,self.accept\_frp\_connection)

self.frp\_cmd\_conn =**None**

self.userConns = []

**def** heartbeat(self):

**while** **True**:

**if** self.frp\_cmd\_conn **is** **not** **None**:

self.frp\_cmd\_conn.send(struct.pack(**'i'**, 1))

time.sleep(9)

*# 收到用户tcp*

**def** accept\_connection(self,sock, mask):

userConn, addr = self.sock.accept()

userConn.setblocking(**True**)

self.userConns.append(userConn)

print(**'收到用户请求'**)

**if** self.frp\_cmd\_conn **is** **None**:

print(**'不存在tcp控制连接'**)

**return**

**try**:

self.frp\_cmd\_conn.send(struct.pack(**'i'**,2)) *#建立新的tcp*

**except** IOError **as** err:

print(err)

**pass**

**def** accept\_frp\_connection(self,sock, mask):

frp\_conn, addr = sock.accept()

frp\_conn.setblocking(**False**)

sel.register(frp\_conn, selectors.EVENT\_READ, self.handle\_controller\_data)

**def** handle\_controller\_data(self,frp\_conn, mask):

**try**:

data = frp\_conn.recv(4)

**if** data:

cmd = struct.unpack(**'i'**,data)[0]

print(**"收到控制报文；"**,cmd)

**if** cmd ==2:

sel.unregister(frp\_conn)

userConn = self.userConns.pop()

frp\_conn.setblocking(**True**)

join(userConn,frp\_conn)

**elif** cmd ==1 **and** self.frp\_cmd\_conn!=frp\_conn:

self.frp\_cmd\_conn = frp\_conn

threading.Thread(target=self.heartbeat).start()

**except** IOError **as** err:

**pass**

**def** run(self):

**while** **True**:

events = sel.select()

**for** key, mask **in** events:

callback = key.data

callback(key.fileobj, mask)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

frpport = int(input(**'请输入frp连接端口:'**))

userport = int(input(**'请输入用户访问端口:'**))

print(**'正在启动FRP服务端...'**)

Frpserver(userport, frpport).start()

print(**'FRP服务端启动成功'**)