好的，我明白了你的意思。你希望客户端只代理那些在代理目录（如 GFWList）中的网站，而**不在代理目录中的网站直接使用客户端本地的网络连接，不经过服务端。**

你希望的效果，应该在**客户端**实现以下逻辑：

1. **PAC 文件解析：** 读取 PAC 文件（或 GFWList），获取需要代理的域名列表。
2. **URL 匹配：** 当客户端收到浏览器/应用程序的请求时，提取 URL，判断该 URL 的域名是否在代理列表中。
3. **选择代理方式：**
    - 如果域名在代理列表中，则将请求转发到服务端进行代理。
    - 如果域名不在代理列表中，则客户端直接向目标服务器发起请求，不经过服务端。

核心思路修改 `client.py`

```
import socket
import threading
import sys
import os
import nacl.secret #实际上用不到了，先import着
import nacl.utils #实际上用不到了，先import着
from nacl.exceptions import CryptoError #实际上用不到了，先import着
import configparser
import urllib.request #引入库用于网络请求


# 新增函数：判断是否需要代理
def should_use_proxy(url, proxy_list_url):

          try:
          # 从网络上加载 blocked list 文件，如果无法加载,也会进入到except里面，确保代码健壮性
              with urllib.request.urlopen(proxy_list_url) as response:
                content = response.read().decode('utf-8')

              need_proxy = False
              for line in content.splitlines():#将block_list按照行分割，进行遍历判断host是否需要代理

                    #进行比较 只需要domain 就可以了吧?
                    #https://github.com/gfwlist/gfwlist 是一个比较出名的项目 里面会有列表，这里为了效果明显，自己写判断逻辑了
                   if is_url_blocked(url, line):
                    need_proxy = True
                    break
                    #     需要用到正则表达式之类的进行判断 是否符合需要翻墙的逻辑

              return need_proxy

          except Exception as e:
                print(f"判断出错了{e}，默认不需要代理")
                return False #代表解析失败,那么默认就是不走代理了
```

```
def is_url_blocked(url, blocked_url):
    """判断 URL 是否被屏蔽"""
    return blocked_url in url

def handle_client(client_socket, client_address, SERVER_HOST,SERVER_PORT,
proxy_list_url):
    """处理客户端连接，根据 PAC 规则选择是否使用代理"""
    print(f"接受来自 {client_address} 的连接")
    try:

        first_package = client_socket.recv(4096) # 接收来自client的数据包,最大为
4096，应该够了

        if not first_package:
             print("Client disconnected")
             return

        def  prase_address(first_package: bytes) -> str:
            try:
            # 将字节数据转换为字符串，假设使用 UTF-8 编码
              http_header_str = first_package.decode('utf-8')

            # 查找Host字段的位置,大小写不敏感
              host_index = http_header_str.lower().find('host:')

              if host_index != -1:
                # 提取Host字段及其值
                 host_line = http_header_str[host_index:].split('\n')[0].strip()
                host_value = host_line.split(':')[1].strip()

                return host_value
              else:
                  return None  # 如果找不到Host字段，返回None
            except Exception as e:
                 print(f"目标地址无法解析{e}")
                 return None


        host = prase_address(first_package)


        if should_use_proxy(host,proxy_list_url):# 使用 should_use_proxy 函数判断
            # 如果需要代理，连接到服务端
            print("需要代理，将请求转发到服务端")

            # 连接到服务端
            server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            server_socket.connect((SERVER_HOST, SERVER_PORT))

            #先发送给服务器，省的重新解析url

            server_socket.sendall(first_package)
```

```
            # 创建线程，用于从客户端转发到服务端
            def forward_client_to_server(client_sock, server_sock):
                try:
                    while True:
                        data = client_sock.recv(4096)
                        if not data:
                            break

                        #发送数据给服务器
                        server_sock.sendall(data)
                except Exception as e:
                    print(f"客户端到服务端转发错误: {e}")
                finally:
                    client_sock.close()
                    server_sock.close()

            # 创建线程，用于从服务端转发到客户端
            def forward_server_to_client(client_sock, server_sock):
                try:
                    while True:
                         data = server_sock.recv(4096)
                         if not data:
                            break

                         client_sock.sendall(data)

                except Exception as e:
                    print(f"服务端到客户端转发错误: {e}")
                finally:
                    client_sock.close()
                    server_sock.close()
            # 分别启动从客户端到目标，以及从目标到客户端的转发线程
            client_to_server_thread =
threading.Thread(target=forward_client_to_server, args=(client_socket,
server_socket))
            server_to_client_thread =
threading.Thread(target=forward_server_to_client,args=(client_socket,
server_socket))
            client_to_server_thread.start()
            server_to_client_thread.start()

            # 等待线程结束
            client_to_server_thread.join()
            server_to_client_thread.join()
        else:
            #如果不需要代理，则在本机直接进行请求
            print("不需要代理，客户端直接发起请求")
            try:
                #从package中解析 目标服务器的地址以及端口号
                 http_header_str = first_package.decode('utf-8')
                 host_line = http_header_str.split('\n')[0].strip()
                 target_host = host_line.split(' ')[1].strip()
```

```
                #从url中解析域名以及端口，不使用try except,解析不出来就走到大的try
except 里面
                target_host_temp = target_host.split(":")
                url =  target_host_temp[0]
                port = int(target_host_temp[1])
                target_address = (url, port) # ip以及 端口号


                target_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

                #和服务端进行连接
                target_socket.connect(target_address)

                target_socket.send(first_package) #首次建立tcp连接的时候 要把这个也
发过去,和服务器建立连接


            except Exception as e:
                print(f"本地client连接出错{e}")
            # 创建线程，用于从客户端转发到目标服务器
            def forward_client_to_target(client_sock, target_sock):
                try:
                    while True:
                        data = client_sock.recv(4096)
                        if not data:
                            break
                        target_sock.sendall(data)
                except Exception as e:
                    print(f"客户端到目标转发错误: {e}")
                finally:
                    client_sock.close()
                    target_sock.close()

            # 创建线程，用于从目标服务器转发到客户端
            def forward_target_to_client(client_sock, target_sock):
                try:
                    while True:
                         data = target_sock.recv(4096)
                         if not data:
                            break

                         client_sock.sendall(data)

                except Exception as e:
                    print(f"服务端到客户端转发错误: {e}")
                finally:
                    client_sock.close()
                    target_sock.close()

            # 分别启动从客户端到目标，以及从目标到客户端的转发线程
            client_to_target_thread =
threading.Thread(target=forward_client_to_target, args=(client_socket,
target_socket))
            target_to_client_thread =
```

```
threading.Thread(target=forward_target_to_client,args=(client_socket,
target_socket))
            client_to_target_thread.start()
            target_to_client_thread.start()

            # 等待线程结束
            client_to_target_thread.join()
            target_to_client_thread.join()



    except Exception as e:
        print(f"处理本地客户端连接时出错: {e}")
    finally:
        client_socket.close()


def main():
    """客户端主函数"""
    config = configparser.ConfigParser()
    config.read("config.ini")

    SERVER_HOST = config["socket"]["host"]
    SERVER_PORT = int(config["socket"]["port"])
    LOCAL_HOST = config["socket"]["local_host"]
    LOCAL_PORT = int( config["socket"]["local_port"])
    proxy_list_url = config["security"]
["proxy_list_url"]#https://github.com/gfwlist/gfwlist


    local_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    local_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    try:
        local_socket.bind((LOCAL_HOST, LOCAL_PORT))
        local_socket.listen(10)
        print(f"客户端监听在 {LOCAL_HOST}:{LOCAL_PORT}")

        while True:
            client_socket, client_address = local_socket.accept()
            client_thread = threading.Thread(target=handle_client, args=
(client_socket, client_address,SERVER_HOST,SERVER_PORT,proxy_list_url))#不要忘记修
改了传递client
            client_thread.start()

    except Exception as e:
        print(f"客户端出错: {e}")
    finally:
        local_socket.close()


if __name__ == "__main__":
    main()
```

在config.ini指定需要跳墙的host服务器

```
[socket]
host = x.x.x.x  #服务端（具有公网 IP 的机器）的公网 IP 地址,如果没有设置,如果设置本机直接连接的话,需要能够访问原网址才行

port = 8080  # 服务端监听端口
local_host = 127.0.0.1 #可以改成0.0.0.0 所有ip都能进行监听

local_port = 9090    # 本地监听端口,客户端通过这个端口和本地进行通信
[security]
#proxy_list_url 代理的地址
#读取url的list 决定要不要走跳板
proxy_list_url = https://mirror.freedif.org/GFWlist/gfwlist.txt #
https://github.com/gfwlist/gfwlist 是一个比较出名的项目
```

总结修改点:

1. 原来的client如果想要指定IP 还是走跳板流程，目前如果匹配proxy的list，是不会经过服务端跳转流程。如果要指定也走 请联系我，我把代码重新发你

代码解释:

- 新增 proxy服务器 代理的IP判断 和端口，如果目标端口 是IP 也可以代理的话会好一些。如果你想要添加这个功能请call me 这个proxy 的 list可以在 https://github.com/gfwlist/gfwlist 上面找到.

步骤:

1.下载github的代码文件,保证 client.py和 config.ini在一个文件下面

验证步骤: 按照提示,访问不被墙的和被墙的网站，看能否正确解析

更完善的代码涉及的技术点:

1. **异步 IO (asyncio):** 为了处理大量并发连接，可以使用 asyncio 来编写非阻塞的代码。
2. **更高效的 PAC 文件解析:** 使用专门的 PAC 文件解析库，例如 `pypac`。

总结. 根据你新的需求.代码全部在client实现.这个server本身 并没有太大的作用.

如果你之前配置好了 `server.py`,需要进行修改 如果你有服务器，`server.py` 作为简单的端口转发： 如果client已经实现了匹配跳墙.那么就作为简单跳转服务

这种架构设计**主要优点:** 逻辑更清晰，**客户端完全掌握代理决策**，服务端只需要提供基本的端口转发功能。** 缺点：**因为代码放在本地client，那么当需要大规模修改 跳墙策略时候。client更新较为麻烦。代码应该尽可能放到server处理

代码写得比较匆忙.可能有格式问题。需要多多包含和理解

请您自行安装这些依赖. 如果代码还有任何格式,运行问题都call me 。 请给我点个赞同