局域網穿透 Intranet Penetration

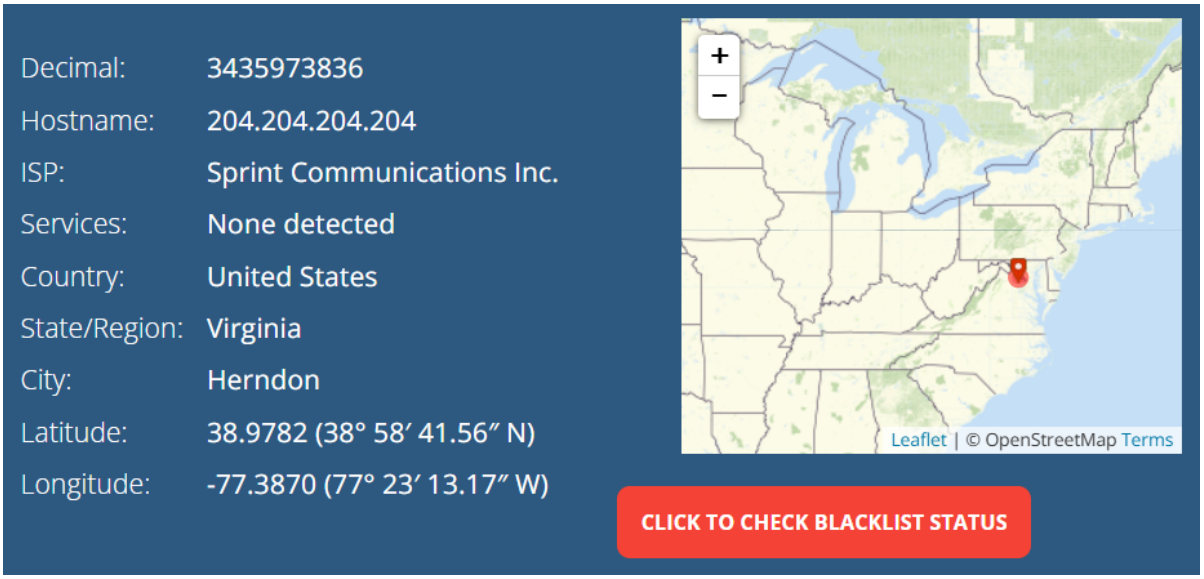
請求的服務器

const char* STUN_SERVER_IP = "stun.l.google.com"; // STUN 伺服器 IP
const int STUN_SERVER_PORT = 19302; // STUN 伺服器端口

最終結果

```
獲取的 IP: 204.204.204.204
獲取的端口: 52428
```

獲取的 IP: 204.204.204.204
獲取的端口: 52428



根據stun協議 ,這個結果和 代碼開始設置 的 STUN_SERVER_IP = "stun.l.google.com" 有什麼關係或邏輯關係

==========

在使用 STUN 協議時，獲取的 IP 地址和端口號與設置的 STUN 伺服器（如 `stun.l.google.com`）之間有特定的邏輯關係。以下是這種關係的解釋：

STUN 協議的工作原理

1. **STUN 伺服器的角色**：
   - STUN（Session Traversal Utilities for NAT）伺服器的主要作用是幫助客戶端在 NAT（網絡地址轉換）環境中獲取其公共 IP 地址和端口號。這對於需要建立點對點連接的應用（如 VoIP 和視頻通話）非常重要。
2. **請求和響應**：
   - 當客戶端向 STUN 伺服器發送請求（例如 Binding Request）時，伺服器會根據請求包中的信息來回應客戶端的公共地址和端口。這些信息是從伺服器的角度來看客戶端的地址。
3. **結果解釋**：
   - 在您的例子中，您獲取到的 IP 地址 `204.204.204.204` 和端口 `52428` 是 STUN 伺服器返回的結果。這表示您的客戶端在 NAT 環境中的公共可見地址是 `204.204.204.204`，且該地址上的端口是 `52428`。

    - 這個 IP 和端口是 STUN 伺服器根據您的請求計算出來的，並不是直接從 `stun.l.google.com` 得到的，而是 STUN 伺服器知道如何將您的請求與其公共接口關聯起來。

總結

- `STUN_SERVER_IP = "stun.l.google.com"` 是您用來發送請求的伺服器地址。
- 獲取的 `204.204.204.204` 和 `52428` 是 STUN 伺服器根據您的請求返回的公共 IP 和端口，這些信息允許其他網絡實體（例如其他客戶端或服務器）能夠通過這個地址來直接與您的客戶端進行通信。

這樣的設計使得在 NAT 環境中，客戶端能夠獲取可用於建立連接的公共地址，從而實現 P2P 通信。

==========

但是本機的的路由器信息如上所示:42.3.90.118, 並不是204.24.204.204？

您觀察到的情況是正常的，因為 STUN 協議的工作原理涉及 NAT（網絡地址轉換）和公共 IP 地址的獲取。以下是一些解釋：

1. 公共 IP 地址與 STUN 伺服器的關係

- **STUN 伺服器的功能**：當您使用 STUN 協議時，您的客戶端會向 STUN 伺服器（如 `stun.l.google.com`）發送請求，伺服器會告訴您它所看到的您的公共 IP 和端口。這是因為 STUN 伺服器位於互聯網上，能夠看到 NAT 設備所映射的地址。
- **獲取的地址**：您獲取到的地址 `204.204.204.204` 是 STUN 伺服器返回的結果，這並不一定是您路由器的 WAN IP。它可能是 NAT 設備映射的地址，或者是 STUN 伺服器所在網絡的某種配置。

2. 本機路由器的信息

- 您提到的路由器信息 `42.3.90.118` 是您本地網絡的公共 IP 地址。這是您路由器向外部互聯網提供的地址。
- 如果您在 NAT 環境中，STUN 伺服器可能會返回一個與您的實際 WAN IP 不同的地址，這可能是由於 NAT 配置或 STUN 伺服器的特定行為。

3. 可能的原因

- **多個 NAT 層**：如果您的網絡中有多個 NAT 層，STUN 伺服器可能會返回一個不同的地址。
- **STUN 伺服器的行為**：有些 STUN 伺服器可能會返回與您的實際公共 IP 不同的地址，具體取決於它們的配置和網絡環境。

總結

在使用 STUN 協議時，獲取到的 IP 地址不一定總是與您的路由器 WAN IP 一致。建議您檢查 NAT 配置和使用其他工具（如 `traceroute` 或其他 STUN 伺服器）來進一步診斷問題。

最後是源代碼:

NatHeartBean.h

```
#pragma once
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cstdint> // 用於 uint16_t, uint32_t 等類型
#include <io.h>
#ifdef _WIN32
#include <winsock2.h>  // Windows socket
#include <ws2tcpip.h>  // Windows TCP/IP
#else
#include <arpa/inet.h> // POSIX socket
#include <unistd.h>    // POSIX standard
#endif

#include "../Common.h"
#include "Config/DeviceConfig.h"

//https://baike.baidu.com/item/stun/3131387?fr=ge_ala 原理
//https://www.bilibili.com/opus/727141412236165127


class NatHeartBean {

public:
    explicit NatHeartBean();
    ~NatHeartBean();
private:
    const char* STUN_SERVER_IP = "stun.l.google.com"; // STUN 伺服器 IP
    const int STUN_SERVER_PORT = 19302; // STUN 伺服器端口
    // STUN 請求的大小
    const int STUN_REQUEST_SIZE = 20; // 根據需要調整大小
public:

    //獲取 IP and PORT
    int get_stun_ip();
    //設置 STUN 請求的內容
    void create_stun_request();
    char stun_request[20];
    int request_length;
};
```

NatHeartBean.cpp

```
#include "NatHeartBean.h"

NatHeartBean::~NatHeartBean()
{
    // 析構函數
}

NatHeartBean::NatHeartBean()
{
```

```
    // 構造函數
}
// ...

int NatHeartBean::get_stun_ip() {

    // 創建 STUN 請求
    create_stun_request();

    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        std::cerr << "Failed to create socket." << std::endl;
        return 1;
    }

    struct addrinfo hints, * res;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET; // IPv4
    hints.ai_socktype = SOCK_DGRAM; // UDP

    // 解析主機名
    if (getaddrinfo(STUN_SERVER_IP, nullptr, &hints, &res) != 0) {
        std::cerr << "Failed to resolve hostname." << std::endl;
        return 1;
    }

    // 將解析的地址設置到 server_addr 中
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(STUN_SERVER_PORT);
    server_addr.sin_addr = ((struct sockaddr_in*)res->ai_addr)->sin_addr;
    // 釋放資源
    freeaddrinfo(res);

    // 將 STUN 伺服器的 IP 和端口設置到 server_addr 中
    inet_pton(AF_INET, STUN_SERVER_IP, &server_addr.sin_addr);

    // 發送請求
    sendto(sockfd, reinterpret_cast<const char*>(stun_request), request_length,
0, (struct sockaddr*)&server_addr, sizeof(server_addr));

    // 接收響應
    char buffer[1024];
    socklen_t addr_len = sizeof(server_addr);

    struct timeval timeout;
    timeout.tv_sec = 5; // 超時時間，單位為秒
    timeout.tv_usec = 0; // 微秒
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&timeout,
sizeof(timeout));
    size_t recv_len = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct
sockaddr*)&server_addr, &addr_len);

    if (recv_len < 0) {
        perror("recvfrom failed");
```

```
        // 處理錯誤
    }

    // 假設 buffer 是接收到的響應
    if (recv_len > 0) {

        // 假設響應中包含了客戶端的 IP 和端口
        char client_ip[INET_ADDRSTRLEN];
        unsigned short client_port;

        // 確保接收到的數據足夠長
        if (recv_len >= 20) { // STUN 消息的最小長度
            // 檢查消息類型和標誌
            uint16_t message_type = (buffer[0] << 8) | buffer[1];
            if (message_type == 0x0101) { // 0x0101 是成功響應的消息類型
                // 提取 Mapped Address
                uint16_t attribute_type = (buffer[28] << 8) | buffer[29]; // 假設
Mapped Address 在第 28 和 29 字節
                uint16_t address_length = (buffer[30] << 8) | buffer[31]; // 地址
長度
                uint32_t client_ip_addr = *(uint32_t*)&buffer[32]; // IP 地址
                unsigned short client_port = ntohs(*(uint16_t*)&buffer[36]); //
端口

                inet_ntop(AF_INET, &client_ip_addr, client_ip,
sizeof(client_ip));
                std::cout << "獲取的 IP: " << client_ip << std::endl;
                std::cout << "獲取的端口: " << client_port << std::endl;
            }
        }
    }

    return 0;
}

// ...

void NatHeartBean::create_stun_request() {
    // 清空請求
    memset(stun_request, 0, STUN_REQUEST_SIZE);

    // 設置請求的標頭
    // stun_request 的結構設置如下：
    // stun_request[0]：版本和類型
    // stun_request[1]：方法碼（Binding Request）
    // stun_request[2] 和 stun_request[3]：消息長度（此處設置為 0，因為沒有額外屬性）
    // stun_request[8]：識別符的開始位置
    //-------------------------------------------------------------------------
-----
     // 設置請求的標頭
    stun_request[0] = 0x00; // 版本和類型
    stun_request[1] = 0x01; // 方法碼（Binding Request）

    // 設置消息長度（這裡為 0，因為沒有額外的屬性）
    stun_request[2] = 0x00;
    stun_request[3] = 0x00;
```

```
    // 設置識別符（隨機生成，這裡簡化處理）
    uint32_t transaction_id = rand(); // 隨機生成一個識別符
    memcpy(&stun_request[8], &transaction_id, sizeof(transaction_id)); // 將識別符放入請求中

    // 設置請求長度
    request_length = STUN_REQUEST_SIZE;
}
```