学　　号 2018116108

密　　级 公开

哈尔滨工程大学本科生毕业论文

面向区块链的 P2P 网络 NAT 穿透技术研究

院（系）名　称：信息与通信工程学院

专　业　名　称：通信工程

学　生　姓　名：蒋佳宁

指　导　教　师：刘春鹏 讲师

校　外　导　师：曹傧 副教授

哈尔滨工程大学

2022 年 **6** 月

面向区块链的 P2P 网络 NAT 穿透技术研究

蒋佳宁

哈尔滨工程大学

学　　号 2018116108

密　　级 公开

面向区块链的 P2P 网络 NAT 穿透技术研究

Research on NAT traversal technology of blockchain-oriented P2P network

学 生 姓 名：蒋佳宁

所 在 学 院：信息与通信工程学院

所 在 专 业：通信工程

指 导 教 师：刘春鹏

职　　称：讲师

所 在 单 位：哈尔滨工程大学

论文提交日期：2022 年 6 月

论文答辩日期：2022 年 6 月

学位授予单位：哈尔滨工程大学

摘　　要

区块链作为一门新型的分布式技术，其在网络层面上是一个 P2P 网络。P2P 网络的一个重要特点是：整个 P2P 网络中所有节点之间的数据收发都不需要经过 P2P 网络以外的其他中间实体，区块链通过 P2P 网络的这种特性实现了所谓的“去中心化”。理论上，P2P 网络可以满足区块链“去中心化”的特性，但在实际的互联网环境中却难以实现，这是由于当前互联网环境中大量存在的 NAT 设备而导致的。隐藏在不同 NAT 设备之下的内网主机之间难以实现 P2P 网络要求的点对点通信，原因在于不同 NAT 设备分配给内网主机的 IP 地址是可重复的，这就导致通过 IP 寻址的 P2P 网络无法通过某一个 IP 地址定位到唯一的主机设备。为了解决当前 NAT 设备对构建 P2P 网络的影响，NAT 穿越技术被提出。

本文的研究内容是 NAT 穿越技术。主要的研究工作如下：

首先将介绍当今市面上主流的几类 NAT 穿越方案并对其各自的优缺点进行分析；然后对在研究领域最受欢迎的 STUN 方案进行详细介绍，包括 STUN 协议的工作原理和各种消息帧结构。

接着将对现存的 STUN 协议进行部分改进，缩短 NAT 穿越需要耗费的时间，减少流量的消耗；在使用 STUN 协议完成 NAT 类型检测之后，根据通信双方 NAT 类型的不同划分不同的穿越场景，采用相对应的方案进行 NAT 穿越。在 UDP 通信场景中以 UDP Hole Punching 穿越方案和端口预测技术为基础，并通过对对称型 NAT 采取更细致的类型划分，优化端口预测技术，进一步降低 NAT 穿越的资源开销；在 TCP 场景下基于 TCP 同时打开的机制，采用一种名为 TCP Simulation Open 的方案来替代 TCP Hole Punching 方案，实现更加快速且高效的 NAT 穿越。

文章的最后给出了整个 NAT 穿越方案 Linux C 语言的实现，包括客户端模块和服务器模块各自程序的实现流程以及使用的消息类型；同时在真实的互联网环境中利用多台云服务器来模拟实现一个 P2P 通信的场景，测试本方案实现 NAT 穿越的可行性并对各场景使用的 NAT 穿越方案消耗的时间进行了统计分析。

关键词：区块链技术; P2P 网络; NAT 技术; STUN 协议; UDP Hole Punching; 端口预测; TCP Hole Punching

ABSTRACT

As a new type of distributed technology, blockchain is essentially a P2P network. An important feature of the P2P network is that the data transmission and reception between all nodes in the entire P2P network does not need to go through other intermediate entities outside the P2P network. In theory, the P2P network can meet the "decentralization" characteristics of the blockchain. But in fact ,this is difficult to achieve. The reason is that the IP addresses under different NAT networks are repeatable, which makes it impossible for the P2P network to locate a unique host device through a certain IP address. To solve this problem, NAT traversal technology is proposed.

The research content of this paper is NAT traversal technology. The main research work is as follows:

First, this paper will introduce several types of mainstream NAT traversal schemes on the market today and analyze their respective advantages and disadvantages; then introduce the most popular STUN scheme in the research field in detail, including the working principle of STUN protocol and various message frames structure.

Second, paper will improve the STUN protocol in terms of time and traffic consumption; According to the different NAT types of the two communication parties, the corresponding scheme is used for NAT traversal. In the UDP communication scenario, based on the UDP Hole Punching traversal scheme and port prediction technology, and by adopting a more detailed type division of symmetric NAT, the port prediction technology is optimized to further reduce the resource overhead of NAT traversal; in the TCP scenario, a A scheme called TCP Simulation Open replaces the TCP Hole Punching scheme to achieve faster and more efficient NAT traversal.

Finally, the specific implementation of the whole NAT traversal scheme is given, the implementation process of the program and the type of messages used; In the actual Internet environment, using cloud server test the feasibility of this solution to achieve NAT traversal, and to perform statistical tests on the time consumed by the NAT traversal solutions.

Keywords: blockchain; P2P network; NAT; STUN; UDP Hole Punching; port prediction;TCP Hole Punching

目　　录

第 1 章 绪论

1.1 研究背景与意义

近些年来，伴随着比特币的出现引出了区块链这一新型技术。区块链技术作为一门新的交叉科学，凭借其独特的“去中心化”思想，吸引了一大批专家学者们开始对它研究。早期的区块链技术应用就是单纯的利用虚拟的电子货币来实现去中心化的交易，但随着以太坊开发出一套可以执行图灵完备的脚本语言的虚拟机，并引入智能合约的概念[1]，在以太坊区块链中可以通过编写智能合约实现各种去中心化应用（Decentralized Application，DApp），这也使得区块链的应用已经不仅仅局限于电子货币的交易，基于区块链的去中心化应用也推动着区块链技术应用于各行各业[2]。

对于区块链的简单定义可以是：一个完全分布式的点对点账本系统，其利用一个特殊的算法，实现对区块内信息生成顺序的协调，并使用加密技术对区块数据进行连接，从而确保了系统的完备性[3]，因此其本质是一套分布式系统。宏观上看，区块链的实现依赖于三项主要的底层技术：共识算法，分布式存储以及 P2P 网络技术。因此一个区块链系统要想正常工作就必须能够实现系统网络中每一个节点的 P2P 通信，然而当前互联网环境中 NAT 设备的广泛存在严重阻碍了这一目标，导致大部分分布式系统只能应用于局域网机房系统。

NAT（Network Address Translatio, 网络地址转换）是 IETF 标准,它是一种把内部私有网络地址翻译成合法网络 IP 地址的技术[4]。NAT 技术的设计初衷主要有两点：一是缓解因互联网快速发展而导致的 IPV4 地址不足问题；二是为了提高内部网络的安全性。这就导致了 NAT 设备只允许内部主机曾经主动访问过的外部主机访问内部主机，对于之前内部主机未曾主动访问过的外部主机，是无法访问内部主机的。所以位于不同 NAT 下的客户端要实现 P2P 通信存在很大困难[5]。根据对比特币和 IPFS 网络的有关统计，当前至少分别有 86.8%和 52.2%的网络主机节点位于 NAT 设备之后[6,7]。因此，设计一种应用层协议解决好 P2P 分布式系统中 NAT 穿越问题，使得位于 NAT 设备之后的内网主机互相通信，是组成更高效、范围更广的分布式系统的关键所在[8]。

IPv6 的提出意味着 IPv4 被淘汰的必然性。在 IPv4 向 IPv6 过渡的阶段，NAT 仍然是一项必不可少的技术手段。因为这种过渡不可能在短时间内完成全网升级，必然是局部

升级，逐渐替换。在两套协议并存的时期，用户和服务资源分布在不同网络之间，跨网访问的需求必须得到满足。可以预见，NAT 在今后相当长的时间内还会大范围的存在，研究如何采取有效的方式越过 NAT 对 P2P 网络连接的阻碍，仍然是相当有必要的。

1.2 国内外研究现状

当前网络环境中的 NAT 设备种类繁多，NAT 的行为也未有明确的标准化规定，导致各大设备生产商难以实现对 NAT 转换的统一。尽管现在已经存在了各种 NAT 穿越技术，但每种方法都难以做到完美。这些存在的 NAT 穿越技术中在效率、成功率、资源占用上都不一样，各自有各自的优缺点和适用范围。不同的 NAT 设备对于数据收发的行为不同，行为越复杂实现 P2P 连接的难度就越大。就目前的状况来看，实现对称型 NAT 的穿越是最为困难的，因为此类设备对于数据包的地址转换方式最为灵活。

总体上看，现存的 NAT 穿越方案可以分为三大类，分别是：更新 NAT 设备、代理转发与直接连接[9]。更新 NAT 设备的方案主要有：ALG(Application Level Gatewa, 应用层网管)、UPnP(Universal Plug and Pla, 通用即插即用)等方案。代理转发的方案主要有：TRUN(Traversal Using Relay NAT, 通过 Relay 方式穿越 NAT)、Full Proxy 等方案。直接连接方案主要是：STUN(Session Traversal Utilities for NAT，NAT 会话穿越应用程序)等方案。

(1) 更新 NAT 设备的方案

此类方案需要对 NAT 设备直接进行更新操作，通过让 NAT 设备下载相应的协议便可以成功实现内网主机的 NAT 穿越，当前区块链技术中的比特币与以太坊都采用的是此类方案中的 UPnP 协议来实现 NAT 的穿越，只要局域网中的路由设备支持 NAT 网关功能、支持 UPnP 协议，即可将你的区块链节点自动映射到公网上[9]。但要实现此类方案对于整体的网络环境是有要求的，它要求从内网主机到达公网环境的所有 NAT 设备都必须支持 UPnP 协议[10]，而要实现这一点通常是比较困难的，这主要是因为当前中国的互联网环境造成的，通常情况下家庭用户到达公网环境需要经过多层 NAT 设备，而这些 NAT 设备中能够由用户进行升级的只限用户本人拥有的，由 ISP 提供的上层 NAT 设备对于用户来说是没有权限进行修改的。

(2) 代理转发方案

此类方案的特点是在通信的节点双方之间存在一个辅助的设备持续为其提供帮助。比如 TRUN 方案，即采用中继的 NAT 穿越方案[11]，需要进行通信的两个节点首先与位于公网环境的 TURN 服务器进行通信，TURN 服务器会为这两个通信节点分别分配一个

通信端口，在接下来的通信过程中，由 TURN 服务器完成对这两个节点间数据的中转，即所有内网主机发送的数据都需要借由 TRUN 服务器实现中继。尽管此类方案成功率是 100%，而且同时支持 UDP 协议和 TCP 协议，但其本身对 TURN 服务器的依赖性过大，导致当数据量变得很大时，TURN 服务器很容易成为网络瓶颈，同时由于报文都必须经过 TURN 服务器转发，增大了报文的延迟和丢包的可能性[12]；与此相比，Full Proxy 采取了另一种思路[12]，这种方案是在内网主机和 NAT 设备之间增加一个 Proxy 设备，这个设备负责解析内网主机产生的信令流与业务数据流，它会将信息中的私有地址改为 Proxy 设备的公网 IP 地址和端口，然后将修改后的消息包发送给 NAT 设备并最终到达对方节点。当 NAT 设备接收到对方的回复之后，由于其下的 Proxy 设备地址是公网地址，因此 NAT 设备知道将回应数据包转发给 Proxy 设备，最后由 Proxy 设备再将数据包转发给对于的内网主机。在 Full Proxy 方案下，不需要对 NAT 设备进行任何改动。但是由于所有的数据包都必须经过 Proxy 设备进行修改后的转发，因此会对 Proxy 设备产生较大的负载，与 TURN 方式相同，该方案也会增加包的延时和丢包的可能性，造成网络瓶颈[8]。

(3) 直接连接方案

直接连接方案以 STUN 方案为主，最早的 STUN 协议由 IETF RFC 3489 定义，全称为 Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)[13]，IETF 为其设立的定位是一套实现 UDP 环境下 NAT 穿越的综合方案。但随着 NAT 设备的逐渐复杂，后续新的 STUN 协议由 IETF RFC 5389 定义，全称为 Session Traversal Utilities for NAT[24]，此时的 STUN 协议仅仅是一种检测 NAT 行为的通用工具。它允许客户端判断自己是否处在 NAT 设备之后，获取映射到公网上的地址信息，检测 NAT 设备的行为，并利用以上信息辅助实现 NAT 穿越。这两种 RFC 文档中规定的 STUN 协议对于 NAT 类型的划分存在些许不同，但具体的实现原理没有变。

STUN 协议特点是利用 NAT 设备对外访问时会留下映射记录的特点，实现不依赖于第三方的双方节点的直接通信，同时也无需对 NAT 设备进行升级改造。由于产生 NAT 映射记录是所有 NAT 都具有的共性，因此直接连接方案可以有效解决多层 NAT 的穿越，虽然此种方案在成功率上相较于以上两种方案稍显劣势，但是其解决方案成本低，占用资源少，因此成为了最受欢迎的一种方案，这也是本设计选取以 STUN 方案实现 NAT 穿越的最主要原因。同时针对于 STUN 协议也不断有学者提出其改进方案，促进了其发展，使得其在泛用性、成功率等方面不断提升。目前针对于此方案的研究的主要方向有：对

称型 NAT 的穿越方案[14-16]、TCP 环境下的 NAT 穿越[9,17-18]、对整体程序的资源优化改善以及结合具体应用场景下的 NAT 穿越[19-21]等方面。

目前随着 STUN 协议的不断发展，越来越多的 NAT 设备制造厂商也在遵守这两种协议下开发其 NAT 产品，使得此种方案的前景大好[22]。

1.3 本文的主要研究内容与结构安排

本文将围绕基于直接连接的 STUN 方案，实现网络中对等节点在 UDP 和 TCP 环境下的 P2P 通信，具体的研究工作如下：

1. 详细调查 P2P 网络技术，了解不同类型的 P2P 网络类型以及 P2P 网络的发展历史；深入学习 NAT 技术，包括其概念、分类、实现的功能；最后具体分析 NAT 技术给 P2P 网络的构建带来的问题。

2.根据 STUN 协议对 NAT 设备分类的不同，确定 NAT 穿越的不同场景并分别采取对应的 NAT 穿越方案，实现不同内网下主机的 UDP 通信和 TCP 通信。

3.根据所设计的理论方案，给出具体的系统框架和模块，包括代码实现和流程图。

4.搭建实验环境，针对最终实现的整个系统进行实际网络测试，通过对各阶段测试的结果进行分析，判断本方案是否有效的实现了 NAT 穿越的目的。

根据本文的主要研究内容将整个论文分为六章，具体的结构安排如下：

第一章是绪论。包含了本文的研究背景，在此基础之上介绍 NAT 穿越技术的研究现状，包括当前比较常用的几类 NAT 穿越方案，并介绍其各自的优缺点，在此基础上给出了本设计选了 STUN 方案的原因。

第二章是基础知识。将本文涉及到的 P2P 网络技术、NAT 技术以及 STUN 协议进行具体介绍，重点分析 NAT 技术为 P2P 技术带来的影响。

第三章对整个 NAT 穿越方案的理论设计进行介绍。将整个方案分为两阶段，第一阶段进行 NAT 类型检测；第二阶段进行 NAT 设备穿越，分为 UDP 环境下和 TCP 环境下的 NAT 穿越。

第四章介绍方案的系统框架和模块实现。先从整体上分析整体系统框架，接着分别从服务器和客户端角度进行模块化功能介绍。

第五章是系统测试，利用云服务器和虚拟机搭建测试环境，对本文实现的 NAT 穿越方案进行测试验证。

最后，对本文的工作进行总结，进一步分析了方案的不足之处，并展望了未来的相关工作。

第 2 章 相关基础知识介绍

2.1 P2P 网络技术

2.1.1 P2P 网络技术概述

在 P2P 网络中，网络的每一个用户我们称为一个对等节点，这些对等节点之间可以相互通信，并且能够根据需求共享网络中其他节点的部分资源(比如 CPU 算力、网络带宽以及存储空间等)。与传统的 C/S 和 B/S 网络不同，P2P 网络中的节点在访问网络中其它节点时，数据的传输往往是点对点直接完成的，而不需要第三方为其提供中继，更重要的一点是：在 P2P 网络中的每一个对等节点不仅可以申请获取对方的资源，而且也可以作为对方节点的资源提供者。

人们常常将 P2P 狭义的理解为 P2P 文件共享和内容分发，在这个时候 P2P 就是一个下载用的术语，它的具体含义是当你在下载某一个文件资源作为资源接收者的同时，还会将下载完成的资源进行上传成为资源的发送者，这种特殊的下载方式带来的结果便是整个网络中对某一资源下载的用户数目越多，其他新用户下载该资源时速度就越快。但实际上，P2P 的应用范围并不只局限于这一方面，在实时通信、网络游戏、协同工作等各种应用领域我们都可以看到 P2P 技术的身影，如今火热的区块链其底层的网络也正是 P2P 网络技术。

2.1.2 P2P 网络的特点

与传统的 C/S 网络模式相比，P2P 网络模式弱化了服务器的角色。在 C/S 模式中，所有网络客户端之间的资源交互都必须通过一个公共的服务器进行处理、中转，这样一来，担任网络中心角色的服务器很容易成为整个系统的性能瓶颈。当服务器无法满足逐渐增多的客户端请求时，就可能导致系统崩溃和瘫痪。同时由于服务器的并发处理能力有限，导致它无法对每一个客户端的请求做到及时的回复，带来的后果就是某些客户端可能会产生比较大的网络延迟，这对于用户的体验是相当糟糕的。

为了解决单台服务器发生负载过重问题，人们提出了负载均衡方案，其简单原理是在客户端与服务器集群之间新增一个负载均衡服务器的角色。每当客户端需要向服务器发送数据时，其请求都会首先被送达负载均衡服务器，负载均衡服务器根据预设的服务

器权重(服务器集群中根据每个服务器的性能为其设置了不同的权重)将数据包进行合理地转发，使得集群中性能高的服务器处理更多的客户端请求，性能较低的服务器不会因负载过重而崩溃。

尽管负载均衡方案是一个比较不错的选择，但却没有从根本上解决问题。一方面整个网络依旧是以整个服务器集群作为核心，而这个服务器集群的处理能力也是有上限的，一旦用户流量突破了这个上限，整个服务器集群就会崩溃。与此相比，P2P 网络中没有中心化的服务器，不存在系统瓶颈，每个节点既当客户端又当服务器，整个网络的处理能力与负载要求能够达到一种动态的平衡[12]。当然，如今更加受欢迎的方案是将 P2P 网络技术与负载均衡技术相结合，取长补短，构建一个性能更加优异的网络，具体分析 P2P 网络技术的特点主要分为以下几类[12]：

1、扩展性强。在一个 P2P 网络中，随着用户节点数目的增多，网络中对于资源的请求也就越来越多，但相应的，随着节点数目的增多，整个网络的处理能力也会不断增强。不断增长的处理能力能够很好的平衡用户需求的增加，这种特性使得 P2P 网络的可扩展性在理论上是趋于无穷的

2、系统鲁棒性强。在 P2P 网络中，没有一个节点像 C/S 架构中服务器一样是必不可少的。P2P 网络通常都是以自组织的方式建立起来的，并允许节点自由地加入和离开。即使部分节点因为某些原因被动地从网络中断开，也不会导致整个网络的瘫痪，P2P 网络还会自动调整网络整体的拓扑结构。

3、传输私密性高。在 P2P 网络中，信息的交互完全由收发的双方节点完成而无需经过第三方(如服务器)的集中处理，这样一来用户节点的隐私数据被泄露的可能性就大大降低了。

4、高性能。P2P 网络可以合理地利用整个网络中大量闲置的用户节点资源。具体的，P2P 网络可以将某一节点的计算请求或资源请求分发至所有的用户节点上，接着利用这些节点的空闲计算能力、带宽共同完成该节点的请求。这种协同工作特性使得 P2P 网络中的用户节点可以获得一种与高性能服务器直接连接相媲美的体验。

2.1.3 P2P 网络结构

P2P 网络按照其组网方式可以分为三大类：集中目录式 P2P 网络、纯 P2P 网络和混合式 P2P 网络，其中纯 P2P 网络又可分为纯 P2P 非结构化网络和纯 P2P 结构化网络[12]。

1、集中目录式 P2P 网络

集中目录式 P2P 网络是最早出现并投入使用的 P2P 网络结构。由于该类型的 P2P 网

络需要一个中心服务器对网络中所有的主机节点进行管理，因此本质上没有抛弃中心化的特点，因此也被称为非纯粹的 P2P 网络。

集中目录式 P2P 网络虽然需要一个中心服务器对所有网络节点管理，但它并非像传统的 C/S 架构中服务器一样，它既不负责存储资源信息，也不负责对所有的节点数据进行中转。在该类型的 P2P 网络中，中心服务器只负责登记并存储整个 P2P 网络中所有节点的索引信息，包括节点的地址、用户名、存储资源的目录等信息，所有的资源数据由网络中的节点进行存储；另外关键的一点是：中心服务器与所有节点之间、节点与节点之间都可以实现双向的自由通信。

从结构上看，集中目录式 P2P 网络采用星型拓扑结构，区域内的所有 P2P 节点都与中央服务器相连接，中心服务器负责记录并存储各个 P2P 节点可供分享的资源列表。网络中的用户节点可以向中心服务器发送对某资源的查询请求，服务器根据自身存储的索引信息找到存储该资源的目标节点，并将其地址返回给该查询用户。之后该用户节点可以与该目标节点建立连接获取想要的资源。

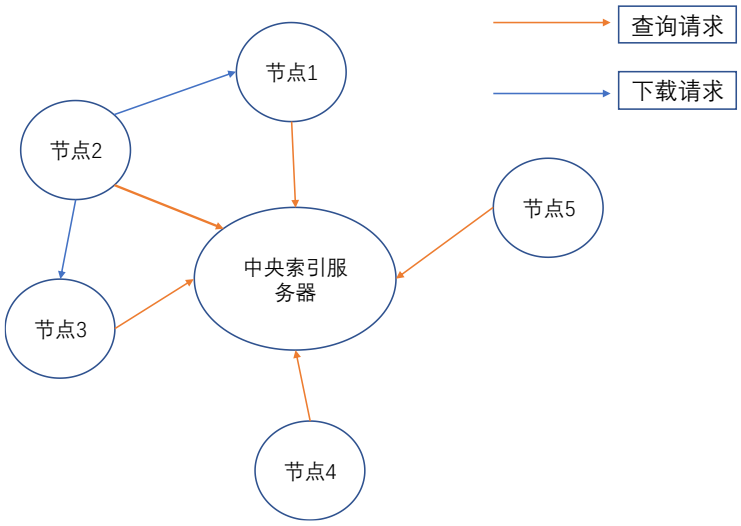

图 2.1 集中式 P2P 网络模型

2、纯 P2P 网络

在一个纯 P2P 网络中，每一个用户节点既是客户端也是服务端，所有的节点拥有相同的地位。每一个节点都维护着一个邻居列表，整个网络中的节点通过与其邻居节点的通信完成数据的流通，从而完全摒弃了服务器的存在。

纯 P2P 网络根据网络的拓扑结构可以进一步分为非结构覆盖网络(Unstructured Overlay Network)和结构覆盖网络(Structured Overlay Network)两种[12]，尽管两者都属于是纯 P2P 网络，但在组网方式上却具有很大差别。

(1) 纯 P2P 非结构化网络

纯 P2P 非结构化网络因为其在进行资源查询时普遍采用泛洪方式因此也被称为广播式的 P2P 网络，网络中所有节点的资源索引和共享都是通过向邻居列表中所有的邻居节点进行广播而完成的。

当某个用户节点连接到整个 P2P 网络中时，它会和与自己在逻辑上相连的其他节点形成从端到端的直接联系，并在最后构成了一个逻辑上覆盖的网络。因为在整个网络中缺乏有效管理者的存在，所以各个节点之间在选择相邻节点时是完全自由的，甚至说逻辑上相邻的两个节点与其实际的物理地址都可能相距甚远;但同时也因为缺乏目录索引，所以网络中任意一个节点对整个网络的认识都仅局限于其在邻居列表上出现的节点。

当某一个节点希望与网络中另一个节点建立连接时，它将会发送一个查询请求广播到所有的邻居节点上，邻居节点会尝试完成这个查询请求，如果不能完成就会以相同的方式进行广播，这种广播行为会不断持续，直到两种情况的发生：一是目标节点的位置被查询成功，二是查询达到了最大步数(TTL)。

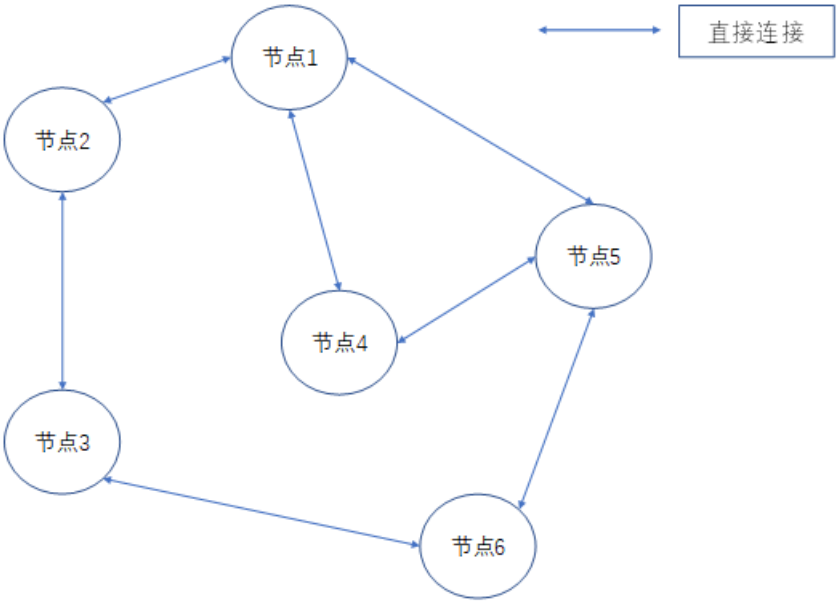

图 2.2 纯 P2P 非结构化网络模型

(2) 纯 P2P 结构化网络

纯 P2P 结构化网络提出的主要目的，是希望改变非结构化网络存在的各种弊端。在结构化的 P2P 网络中，由于各个节点之间在进行邻居选择上是有规则的，因此资源不再可以随意地存放在整个网络中的结点上，而是会以一个更便于检索的方法存放在适当的节点上。

这种结构化的组织方式通过分布式散列表(DHT)技术实现，在非结构化 P2P 网络的

基础上通过分布式散列函数(也称分布式哈希函数)并结合路由算法实现网络中节点之间资源的查找。

我们可以对 DHT 的思想进行简单理解：整个 P2P 网络维护着一张巨大的文件索引表，网络中每个节点根据自身性能的不同分别维护整个索引表的一部分内容。整个索引表由一条条的文件索引条目组成，每条索引条目分为两部分：key 和 value，其中 key 称为关键字，它是某一资源文件的文件名(或者其他可以唯一标识改文件的属性)的哈希值；value 是在网络中存储改文件的节点的地址信息(通常为 IP 地址)。每当网络中某一节点需要获取网络中某一资源时，只要通过哈希函数获取该资源文件的 key 值，然后再使用该 key 值在文件索引表中进行查询就可以快速定位到该资源的具体位置。

3、混合式 P2P 网络

混合式 P2P 网络对集中目录式和纯 P2P 网络进行了结合，它综合考虑了这两种 P2P 网络的优缺点，实现了“部分去中心化”的网络结构。混合式 P2P 网络根据各节点性能(通常考虑 CPU 性能、存储性能、网络带宽等)的不同将其分为普通节点和超级节点两大类。当这两类节点作为资源的获取者时，会被视为对等节点具有相同的地位。不同的是，超级节点会具有一些特殊功能：每一个超级节点将存储整个 P2P 网络中某一区域的普通节点和与其相邻的其他超级节点的信息，所有的资源查询请求经过的路径除了起点不会存在任何普通节点，超级节点将负责转发节点们的资源查询请求，然后进行目标节点的查询或查询请求的转发。

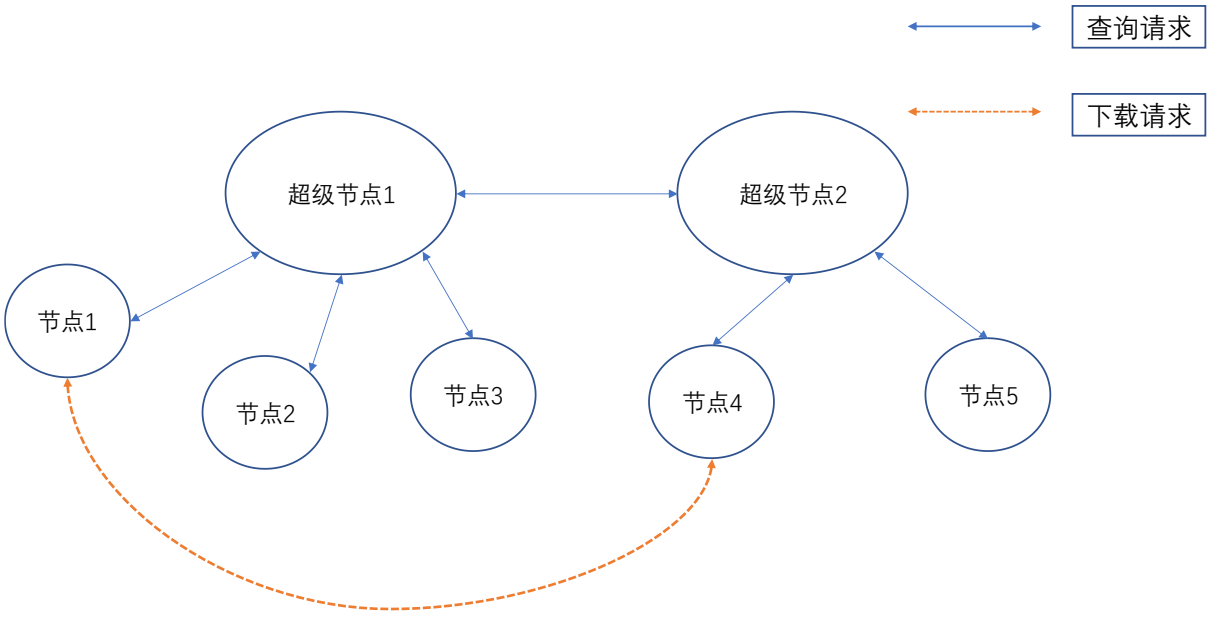

图 2.3 混合式 P2P 网络模型

2.2 NAT 技术

2.2.1 NAT 技术概述

NAT（Network Address Translation）是 IETF 标准,它是一种把内部主机的私有网络地址转换为互联网公网 IP 地址的技术。NAT 技术隐藏了其下的所有内部主机信息, 使外界无法直接访问内部主机，从而实现了一种保护功能; 更重要的一点是在 IPv4 地址严重不足的情况下解决了互联网时代用户们的上网问题。

IANA 组织在 RFC1918 中规定了 3 类网络私有地址，这 3 类地址不会在因特网上分配，仅可在局域网内部使用。由于只能在局域网中使用，因此不同局域网中可以使用相同的地址。

表 2.1 私有 IP 地址空间

私有 IP 地址空间	A 类	10.0.0.0 ~ 10.255.255.255
	B 类	172.16.0.0 ~ 172.31.255.255
	C 类	192.168.0.0 ~ 192.168.255.255

2.2.2 NAT 的工作原理

NAT 的存在使得局域网中的大量主机可以共享使用少量的公网 IP 地址，当内网中的主机通过 NAT 设备向外界发送数据时，NAT 设备会将该主机的内网地址转换为可被外界识别的公网地址。同时 NAT 设备还留下一条该会话的 NAT 映射记录，一旦外网设备对此内网主机进行了回复，NAT 设备能够根据此映射记录将其转发给该内网主机。下图 2.4 详细展示了 NAT 的工作流程：

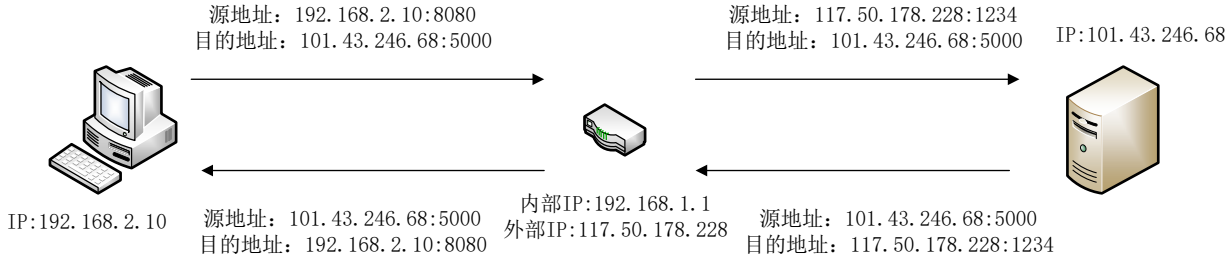

图 2.4 NAT 的工作原理

在上图 2.4 中，路由器担任 NAT 转换设备的角色，当路由器下的内网主机 IP:192.168.2.10 试图向公网服务器 IP:101.43.246.68 发送消息时，消息首先会被转发到作为网管的路由器上。路由器将修改该数据包的源地址信息，同时在自身的 NAT 转换表中增加此次通信对于的会话记录；当路由器收到服务器回复的数据包时，首先会检索 NAT

转化表，查询目的地址 117.50.178.228:1234 是否存在相应的映射，如果没有将丢弃该数据包；如果有则修改数据包的目的地址并向内网转发这一数据包。

通过实现以上功能，使得具有内网 IP 地址的主机在 NAT 设备的协助下也可以自由的访问外网地址。同时网络上除 NAT 设备以外的任何设备都可以按照以往的规范继续工作，这样就以最小的改动解决了 IPV4 地址不足的问题。

2.2.3 NAT 的分类

按照采取分类特征的不同，NAT 有多种分类方式。这里我们按照 RFC 3489 所采用的 NAT 设备在进行地址转换时的规则对 NAT 设备进行分类，可以分为以下四种类型[5]，四类 NAT 之间的关系如图 2.5 所示。

1、完全锥型 NAT(Full Cone NAT)

2、限制锥型 NAT(Restricted Cone NAT)

3、端口限制锥型 NAT(Port Restricted Cone NAT)

4、对称式 NAT(Symmetric NAT)

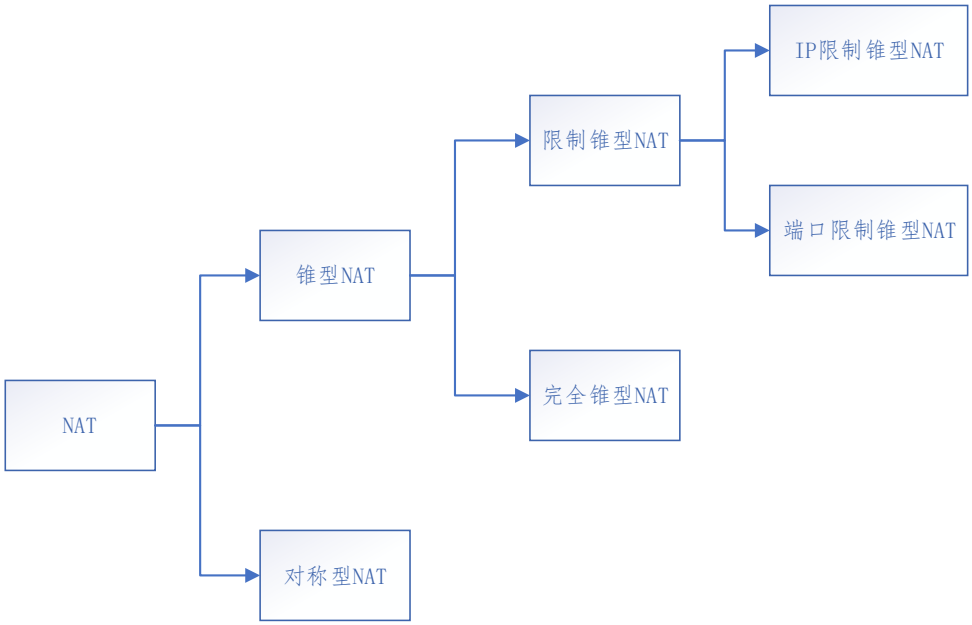

图 2.5　NAT 设备的分类

为了便于说明，下面将内网主机地址规定为(in_IP : in_Port)，将经过 NAT 映射后的地址规定为(ex_IP : ex_Port)，被访问的公网服务器的地址规定为(pub_IP : pub_Port)。

1、完全锥型 NAT (Full cone NAT) :

一旦内网主机(in_IP : in_Port)经过映射地址变为(ex_IP : ex_Port)之后，向服务器(pub_IP : pub_Port)发送了一次消息，那么在 NAT 的映射记录存在期间，下一次(in_IP :

in_Port)向公网环境中发送消息，其数据包在经过 NAT 设备时都会被重新映射为(ex_IP : ex_Port)。同时所有网络主机都可以通过给(ex_IP : ex_Port)发送数据包而被转发到内网主机(in_IP : in_Port)。

总结：同一个内部地址(in_IP : in_Port)只会映射相同的外部地址(ex_IP : ex_Port)，映射完成后，NAT 设备对外来数据包的 IP 和 Port 都不进行过滤，而是将所有发送至(ex_IP : ex_Port)的数据包转发至(in_IP : in_Port)。

2、IP 限制锥型 NAT（Address-Restricted cone NAT）

一旦内网主机(in_IP : in_Port)经过映射地址变为(ex_IP : ex_Port)之后，向服务器(pub_IP : pub_Port)发送了一次消息，那么在 NAT 的映射记录存在期间，下一次(in_IP : in_Port)向公网环境中发送消息，其数据包在经过 NAT 设备时都会被重新映射为(ex_IP : ex_Port)。与完全锥型 NAT 不同的是，只有源 IP 地址为 pub_IP 的设备发送的数据包才会被 NAT 设备接收，IP 地址不符的数据包将会被 NAT 设备丢弃，NAT 设备在收到这些地址为（pub_IP：xxx）的数据包之后，会将其转发至内网主机(in_IP : in_Port)。

总结：同一个内部地址(in_IP : in_Port)只会映射相同的外部地址(ex_IP : ex_Port)，映射完成后，NAT 设备对外来数据包的源 IP 进行过滤，将所有符合 IP 过滤要求的数据包转发至 (in_IP : in_Port)。

3、端口限制锥型 NAT（Port-Restricted cone NAT）：

一旦内网主机(in_IP : in_Port)经过映射地址变为(ex_IP : ex_Port)之后，向服务器(pub_IP : pub_Port)发送了一次消息，那么在 NAT 的映射记录存在期间，下一次(in_IP : in_Port)向公网环境中发送消息，其数据包在经过 NAT 设备时都会被重新映射为(ex_IP : ex_Port)。相比于 IP 限制锥型 NAT，端口限制锥型 NAT 过滤规则更加严格，只有源地址为(pub_IP : pub_Port)的数据包才会被 NAT 设备接收，IP 地址与端口号任意一项不符合规则都会被 NAT 设备丢弃，NAT 设备在接收到这些来自(pub_IP : pub_Port)的数据包之后会将其转发至内网主机(in_IP : in_Port)。

总结：同一个内部地址(in_IP : in_Port)只会映射相同的外部地址(ex_IP : ex_Port)，映射完成后，NAT 设备对外来数据包的源 IP 和源端口号同时进行过滤，只有同时符合这两项过滤要求的数据包才会被转发至 (in_IP : in_Port)。

4、对称型 NAT（Symmetric NAT）：

对称型 NAT 是最为严格，也是最为安全的一类 NAT，这表现在它不仅拥有向端口限制锥型 NAT 一样严格的过滤规则而且在 NAT 映射上也进行了随机化：每一个来自相

同内部 IP 与端口，到一个特定目的地地址和端口的请求，都将映射到一个单独的外部 IP 地址和端口上。即使说，同一台内网主机(in_IP : in_Port)分别向公网服务器的两个端口发送了一条消息，即目的地址分别为(pub_IP : pub_Port_1)和(pub_IP : pub_Port_2)，由于两次目的地址的端口号不同，在进行 NAT 映射时分别映射为(ex_IP_1 : ex_Port_1)和(ex_IP_2 : ex_Port_2)，通常情况下 ex_IP_1 = ex_IP_2，但 ex_Port_1 与 ex_Port_2 绝对不相等。

映射后的(ex_IP_1 : ex_Port_1)和(ex_IP_2 : ex_Port_2)分别只接收来自于(pub_IP : pub_Port_1)和(pub_IP : pub_Port_2)的数据包，既限制服务器数据包的源 IP 地址，也限制服务器数据包的源端口号。这就严格的特性导致了对称型 NAT 在没有其他辅助手段的情况下只能实现一对一的数据通信。

总结：同一个内部地址(in_IP ： in_Port)与不同的外部地址进行会话时会映射为不同的外部地址(ex_IP ： ex_Port)，映射完成后，NAT 设备对外来数据包的源 IP 和源端口号同时进行过滤，只有同时符合这两项过滤要求的数据包才会被转发至 (in_IP ： in_Port)。

2.3 NAT 对 P2P 技术的具体影响

NAT 设备将屏蔽所有未经授权的来自公网的数据包。所谓“未经授权”是指没有内网主机发起连接在先而“不请自到”的数据包，NAT 对这样的数据包直接抛弃，因此导致了不同 NAT 设备后的主机无法进行通信的问题，因为双方都无法向对方发起连接，所有连接请求包以及通信数据包都将被对方的 NAT 抛弃[23]。

NAT 设备这种行为严重限制了 P2P 网络中节点的连接需要。因为在一个 P2P 网络中，无论是内网环境下的主机节点还是公网环境下的主机节点，都认为他们是对等的，这种对等性要求每一个节点既能够作为数据的发送方，也可以作为数据的接收方。但是由于 NAT 的屏蔽机制，内网下的主机节点无法主动成为数据的接收方。而且如果收发的双方节点都各自位于不同的内网中无论是数据的发送还是接受都是无法完成的，节点之间的交流被完全阻断。

2.4 STUN 协议介绍

本设计综合考虑这两种 RFC 文档中对 STUN 协议的定位与实现复杂度，以 RFC 3489 为基础，但仅将其作为 NAT 类型检测的工具，通过利用 STUN 协议，一个客户端能够探测自己是否处在 NAT 之后，如果处于 NAT 之后，则可以检测出此 NAT 的类型，并探测

出此 NAT 为其分配的公网 IP 和端口号。利用得到的这些信息再采取相应的 NAT 穿越方法使处于 NAT 设备之后的主机之间建立连接，进行通信[25]。

STUN 协议在 NAT 类型检测上的实现原理是首先由内网主机用户发送请求给公网服务器，当服务器收到客户端的消息请求之后，将接收数据包中的源地址信息加入到响应数据包中返回给客户端。当客户端接收到响应数据包之后，即可获得其自身所经过的 NAT 设备为其映射的公网 IP 地址和端口。之后其他客户端就在服务器的帮助下获取到此对外的 IP 地址信息，并用此信息来与此客户端建立连接通信[25]。

STUN 消息是使用大端字节流编码的 TLV(type-length-value)。 STUN 消息都是由消息头与载核数据构成的。

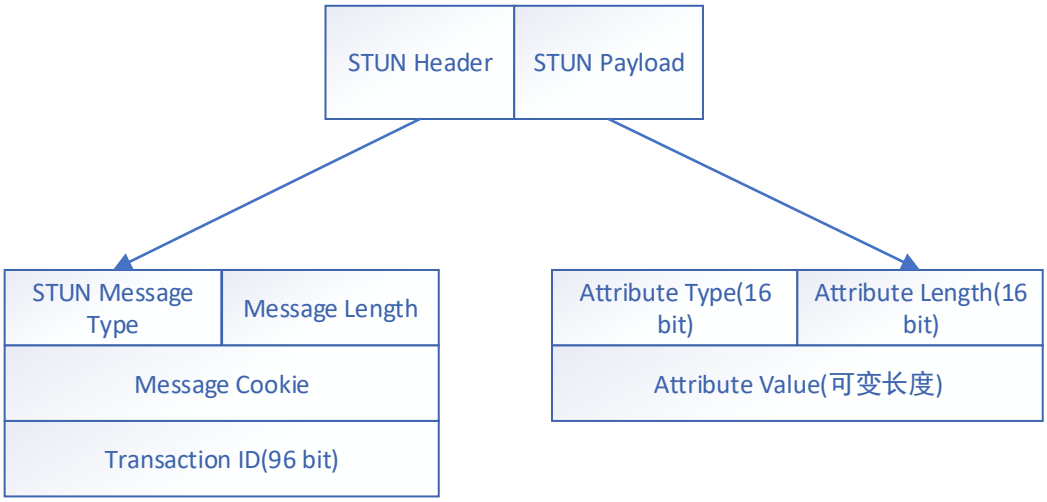

图 2.6 STUN 协议数据帧结构

1、STUN 消息头介绍

对于客户端，STUN 协议主要定义了两种类型的消息：分别为捆绑请求(Binding Requests)和共享私密请求(Shared Secret Requests)。其中捆绑请求用于 UDP 数据包的请求，与服务器的响应消息相对应，用来判断客户端与服务器之间是否有 NAT 设备，如果有则分辨其类型。而共享私密请求被使用于 TCP 的连接，客户端向服务器发送请求，服务器接收到消息之后，返回一个临时的用户名和密码，用来检测消息的安全性和完整性。

对于服务器，STUN 主要定义并使用以下的四种类型的消息:捆绑响应(Binding Responses)、捆绑错误响应(Binding Error Responses)、共享私密响应(Shared Secret Requests)和共享私密错误响应(Shared Secret Error Responses)，这四种消息类型分别与客户端的消息类型相对应。服务器返回的捆绑响应、捆绑错误响应对应于客户端的捆绑请

求，配合客户端分辨它们之间是否有 NAT 设备以及 NAT 设备的类型。而服务器返回的共享私密响应和共享私密错误响应对应于客户端的共享私密请求，判断是否给客户端发送临时用户名和密码。

2、STUN 消息载荷介绍

在 STUN 消息的 Payload 中包含了该消息的具体属性，STUN 客户端与服务器通过分析属性字段可以进一步获知该消息的作用。STUN 消息可选的属性共有 11 种，一条 STUN 消息的载荷部分可以包含多条属性字段。下面介绍几种常用的属性：

(1) MAPPED-ADDRESS：

此种属性只会存在于服务器的响应消息中，用于指示此消息中含有客户端经过 NAT 映射后的公网地址信息，具体的地址信息内容将存放于 Attribute value 字段中，该字段由 8 位地址协议簇、16 位端口号和 32 位 IP 地址构成。

图 2.7 MAPPED-ADDRESS 的 Attribute value 字段

(2) CHANGE-REQUEST：

此属性只会存在于客户端的请求消息中，用于请求客户端采用不同的 IP 地址或端口号进行对自身请求的回复，该属性的 value 字段总共 32 位，但只有两位实际起作用，A 和 B。

图 2.8 CHANGE-REQUEST 的 Attribute value 字段

其中，A 是“change IP”标志。如果被置 1，则表示请求服务器更换一个与客户端目的 IP 地址不同的另一个 IP 进行回复；B 是“change port”标志，如果被置 1，则表示请求服务器更换一个与客户端目的端口不同的另一个端口进行回复。

(3) CHANGED-ADDRESS：

此种属性只存在于服务器的响应消息中，当服务器收到的请求带有 CHANGE-REQUEST 属性时，服务器将会使用不同的 IP 或者端口回应客户端，此时的 STUN 消息便会携带 CHANGED-ADDRESS 属性，来表示响应发出的 IP 和端口。

2.5 本章小结

本章首先介绍了 P2P 网络技术的基本原理以及其应用场景，介绍了 P2P 模式与传统

的 C/S 模型相比有哪些优点并针对于三大类 P2P 网络模型进行了深入的介绍；接着又针对 NAT 技术进行了深入探讨，包括其工作的原理和分类方式，同时具体分析了 NAT 给 P2P 技术带来的影响。在本章的最后介绍了 STUN 协议的作用以及简单的工作原理。

第 3 章 基于 STUN 的 NAT 穿越方案的设计与改进

本章基于 STUN 协议的 NAT 穿越方案，对整体穿越流程以及相关原理进行介绍。将整个 NAT 穿越过程分为 NAT 类型检测和具体环境下的 NAT 穿越两部分进行分析；同时将针对现有 NAT 穿越方案的不足之处进行分析，并对方案进行相应的改进，从整体上提高 NAT 穿越的成功率、减少穿越过程的时间和资源耗费。

3.1 NAT 类型检测阶段

3.1.1 RFC3489 中的标准 NAT 类型检测方案

RFC3489 标准中给出了对 NAT 类型进行检测的整体步骤，如下图所示。

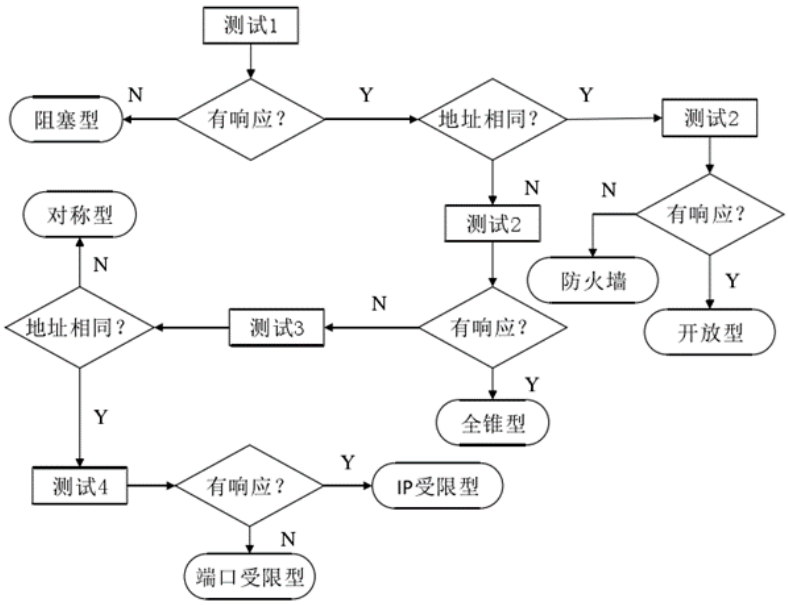

图 3.1　STUN 协议的 NAT 类型探测流程[5]

为了方便说明，下面我们假设位于公网的 STUN 服务器存在两个独立的公网 IP 地址，分别为 IP_S1 和 IP_S2，默认使用的两个端口号分别为 3478 和 3479。

测试 1：客户端向服务器的 IP_S1:3478 发送 Requests 消息，消息载荷包含字段属性 CHANGE-REQUEST，其中 A 和 B 都设置为 0，即要求服务器不做任何改动而是能够以 IP_S1:3478 对客户端的 Requests 直接进行 Responses 回复，其中 Responses 的消息载荷中的 MAPPED-ADDRESS 字段包含了客户端在 NAT 映射之后的公网地址。紧接其后的第一个地址相同判断指的是：客户端将从 STUN 服务器处接收到自身 NAT 映射的公

网 IP 地址与自身 IP 地址进行比较，判断这两个地址是否相同。

测试 2：客户端向服务器的 IP_S1:3478 发送 Requests 消息，消息载荷包含字段属性 CHANGE-REQUEST，其中 A 和 B 都设置为 1，即服务器将使用 IP_S2:3479 对客户端的 Requests 进行 Responses 回复。

测试 3：客户端向服务器的 IP_S2:3479 发送 Requests 消息，消息载荷包含字段属性 CHANGE-REQUEST，其中 A 和 B 都设置为 0，即要求服务器不做任何改动而是能够以 IP_S2:3479 对客户端的 Requests 直接进行 Responses 回复。之后的第二个地址相同判断指的是：客户端将此次从 STUN 服务器处接收到自身 NAT 映射的公网 IP 地址与测试 1 中获得的映射结果进行比较，判断其是否相等。

测试 4：客户端向服务器的 IP_S1:3478 发送 Requests 消息，消息载荷包含字段属性 CHANGE-REQUEST，其中 A 设置为 0，而 B 设置为 1，即服务器将使用 IP_S1:3479 对客户端的 Requests 进行 Responses 回复。

下面我们对 NAT 类型探测的结果进行分析：

如果客户端在测试 1 中没有得到服务器响应，这种情况一般是由于 STUN 服务器没有关闭针对 3478 端口的防火墙，导致客户端无法访问服务器；如果客户端得到了响应而且符合地址相同的判断同时又在测试 2 中能够得到响应，说明客户端本身就位于公网环境中，不存在 NAT 设备。如果客户端不符合第一个地址相同的判断，那么可以说明客户端存在 NAT 设备。若在测试 2 中能够得到响应，说明其 NAT 设备为 Full Cone 类型；如果不能得到响应就继续进行测试 3。在第二处地址相同判断处，若不相同就可以判断处其 NAT 设备为 Symmetric 类型；如果相同继续进行测试 4。如果测试 4 中，客户端能够得到响应说明其 NAT 设备为 IP Restricted Cone 类型；否则为 Port Restricted Cone 类型。至此，我们完成了对 NAT 设备类型的判断。

3.1.2 改进的 NAT 类型检测方案

从整个 RFC3489 中 NAT 类型的探测流程中我们可以看出整个测试流程是串行进行的，也就是说整个测试流程中的每一次测试都是依赖于上一次的测试结果的。我们暂且不考虑程序运行时间，单纯从探测方案逻辑上进行分析完成一次客户端的 NAT 类型的检测需要多长时间。

以测试 1 为例，如果此时 STUN 服务器忘记关闭 3478 端口防火墙，那么客户端是无法得到服务器响应的，但客户端不可能在向服务器发送 Requests 消息之后因为没能得到服务器回复就立即退出检测方案，并判断是服务器没关闭防火墙，因为存在网络延

迟的可能性。在进行具体实现时，我们都会为客户端设置一个时间阈值，如果超过这个阈值客户端仍然没能收到服务器响应，这时候我们才会结束检测并做出相应的结果判断。根据相关文献[5]，这个时间阈值一般设置为 2s。下面我们对整个探测流程进行一个时间消耗统计：

首先我们进行测试 1，如果没有响应就会等待 2s 给出最终结果，如果有响应就立即进行地址判断；此时路线分为两条：①如果 NAT 设备为开放型(无 NAT)且服务器开放 3479 端口就会直接完成整个检测方案，但如果服务器没有开放 3479 端口就需要等待 2s 给出结果；②如果客户端存在 NAT 设备，且为 Full Cone 类型，那么我们就可以立刻完成检测方案，但如果 NAT 设备为其他类型就需要等待 2s，继续进行测试 3；测试 3 中在网络无延迟时可以立即收到服务器回复，然后根据回复进行分析判断是否为 Symmetric 类型，若不是接着继续进行测试 4；测试 4 中，如果 NAT 设备为 IP Restricted Cone 类型，可以立即完成检测方案，但如果为 Port Restricted Cone 类型，此时理论上我们不需要再等待，因为现在只剩下 Port Restricted Cone 型 NAT 这一种可能性，但是我们还需要考虑到另一种可能性，即是由于网络延迟等问题，导致服务器回复的消息存在时延，因此仍需要等待 2s 后再针对这两种存在的可能性做出具体的判断，如果在超出时间阈值后仍然没能得到回复才可以判断此时的 NAT 类型确实为 Port Restricted Cone 型。下面给出一个表格对各种探测结果所需时间进行统计表示。

表 3.1 原 NAT 类型检测方案所需理论时间

NAT 类型	服务器防火墙情况	理论最少所需时间	理论最多所需时间
任意	3478,3479 至少一个未开启	2s	4s
开放型(无 NAT)	3478,3479 都开启	0s	4s
Full Cone 类型	3478,3479 都开启	0s	4s
Symmetric 类型	3478,3479 都开启	2s	6s
IP Restricted Cone 类型	3478,3479 都开启	2s	8s
Port Restricted Cone 类型	3478,3479 都开启	4s	8s

我们可以看到，在最理想的情况下(没有网络延迟)，除了对于 Port Restricted Cone 类型 NAT 检测时间比较长，其他 NAT 类型检测所消耗的时间还是不错的。但是一旦网络延迟比较高，每次客户端发送请求后都需要等待一段时间后才能得到响应，这就会导致整体所需时间增多，上表中第四栏即为每种 NAT 设备在进行类型检测时最多需要耗

费的时间。

根据整个流程我们可以看出，整个方案的时间消耗是累加的，越是排在后面的 NAT 类型(比如两种限制锥型 NAT 和对称型 NAT)需要消耗的时间就越多。而且更重要的一点是：当今市面上的大部分 NAT 设备恰好就是这三种类型中的 Port Restricted Cone 类型和 Symmetric 类型，家庭用 NAT 设备基本上都是 Port Restricted Cone 类型，而公司企业一般使用 Symmetric 类型的 NAT。Full Cone 类型和 IP Restricted Cone 类型基本占很少份额(这是因为这两种类型的 NAT 安全性较低，容易遭受不法分子攻击)。

导致 Symmetric 类型和 Port Restricted Cone 类型 NAT 检测缓慢的原因主要是因为整个探测流程是同步的，都至少需要在测试 2 中等待一次，遇到网络延迟比较大的时候，可能还需要在其它测试(1、3、4)处进行等待，要解决这个问题我们很容易就可以想到使用多线程进行优化，使得流程由同步变异步，大大缩减所需时间。但线程的开启也是需要耗费资源的，而且 NAT 穿越一般是作为大型项目的小部分，一个项目中线程越多，线程之间的切换就需要耗费更多的资源和时间，导致的结果可能是项目整体性能的下降。这里我们从不同的角度出发，考虑一种无须多线程也能实现的异步化：

整个 NAT 方案的实现是由客户端向服务器依次发送 2 次 Request 消息，然后服务器采用不同的网络地址对这 2 次 Request 分别进行 4 次回复，而对 NAT 类型的判断依据是客户端能否在消息发送后收到服务器用不同网络地址的回复。即是说问题的关键是客户端向着服务器某一个网络地址发送消息，消息的内容是指示服务器用指定的网络地址对自身进行回复，最后客户端根据自身能否收到服务器的回复判断自身 NAT 类型。

如此一来，我们便可以采用如下的方案来代替原始的 RFC3489 中的 NAT 类型检测方案：

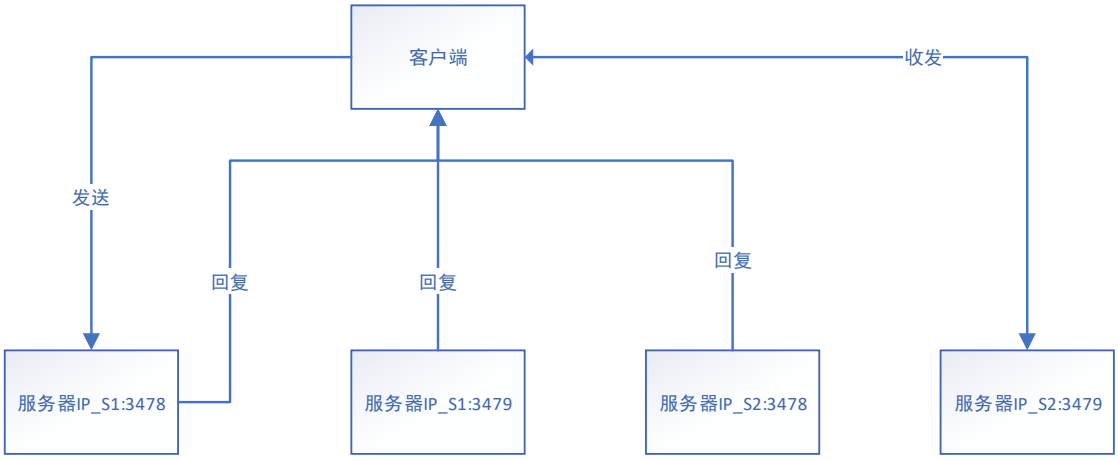

图 3.2 改进的 NAT 类型检测方案流程图

如上图所示，我们让客户端向服务器的 IP_S1:3478 发送消息，指示服务器分别用 IP_S1:3478 、IP_S1:3479 、IP_S2:3478 向客户端进行回复，回复消息的内容包括客户端在 NAT 映射后的网络地址，然后客户端根据能收到的消息的个数对自身的 NAT 设备类型进行判断。由于涉及到对 Symmetric NAT 和 Port Restricted Cone 的区分，还需要客户端同时向 IP_S2:3479 发送消息，并指示服务器用 IP_S2:3479 进行回复。

1、如果客户端对于服务器发送的四条回复都能收到。

此时客户端的 NAT 类型要么是 Full Cone 类型要么是开放型(无 NAT)。原因是：由于能收到服务器用 IP_S1:3478 和 IP_S2:3479 地址进行的回复，说明服务器不存在防火墙问题；由于能收到服务器用 IP_S1:3479 地址进行的回复，说明客户端 NAT 设备对于外来消息包的端口号不敏感，可能为 Full Cone 型或 IP Restricted 型或是无 NAT；最后，由于能收到服务器用 IP_S2:3478 地址进行的回复，说明其本身对外来消息包的 IP 地址不敏感，只能是 Full Cone 型或是无 NAT。接着我们可以将 Response 消息中 NAT 映射的网络地址与客户端自身网络地址进行比较，若相等则为开放型 NAT，否则为 Full Cone 型。

2、如果客户端只能收到来自服务器 IP_S1:3478、IP_S2:3479 和 IP_S1:3479 的回复。

这种情况说明自身的 NAT 对外来消息包的端口号不敏感，但是对 IP 地址敏感，符合此条件的只有 IP Restricted 型的 NAT 设备。

3、如果客户端只能收到来自服务器 IP_S1:3478 和 IP_S2:3479 的回复。

这说明自身的 NAT 不仅对外来消息的 IP 地址敏感，对端口号也同样敏感，而符合此条件的有两种类型的 NAT：Port Restricted Cone 型和 Symmetric 型，此时我们需要采取进一步的判断：客户端需要比较来自于服务器 IP_S1:3478 和 IP_S2:3479 的消息包中两次 NAT 映射后的网络地址是否相同，若相同则为 Port Restricted Cone 型 NAT，否则为 Symmetric 型 NAT。

4、如果客户端无法收到任何服务器消息。

说明服务器未关闭相应端口的防火墙，导致测试失败。

针对以上新方案，我们可以计算其理论所需的时间，如下表 3.2。

表 3.2 改进的 NAT 类型检测方案所需理论时间

NAT 类型	服务器防火墙情况	理论最少所需时间	理论最多所需时间
任意	3478,3479 至少一个未开启	2s	2s

续表 3.2 改进的 NAT 类型检测方案所需理论时间

NAT 类型	服务器防火墙情况	理论最少所需时间	理论最多所需时间
Full Cone 类型	3478,3479 都开启	0s	2s
Symmetric 类型	3478,3479 都开启	2s	2s
IP Restricted Cone 类型	3478,3479 都开启	2s	2s
Port Restricted Cone 类型	3478,3479 都开启	2s	2s

我们将改进前后的两种 NAT 类型检测方案进行比较可以发现：在理想的，没有网络延迟的情况下，新方案只在进行对 Port Restricted Cone 类型的 NAT 进行检测时耗费的时间缩减了一半，其余皆和原方案一致。但是在最坏的情况时下，改进的方案是完全优于原始 RFC3489 方案的，而且从流量耗费上看：原始 RFC3489 方案客户端最多需要进行四次探测包发送；改进后客户端只需要发送两次探测包。从而节省了不必要的开支。

3.2 UDP 环境下的 NAT 穿越

在完成对客户端所属 NAT 设备类型判断完成之后，就可以进行具体的 NAT 穿越了。根据不同的需要，分为 UDP 和 TCP 两种环境下的 NAT 穿越，下面首先介绍 UDP 环境下的 NAT 穿越。

3.2.1 针对不同 NAT 穿越场景的方案设计

假设我们现在有两台网络主机 A 和 B 需要实现 NAT 穿越，根据两者 NAT 所属情况的不同，存在着不同的 NAT 穿越方案，如下图 3.3 所示。

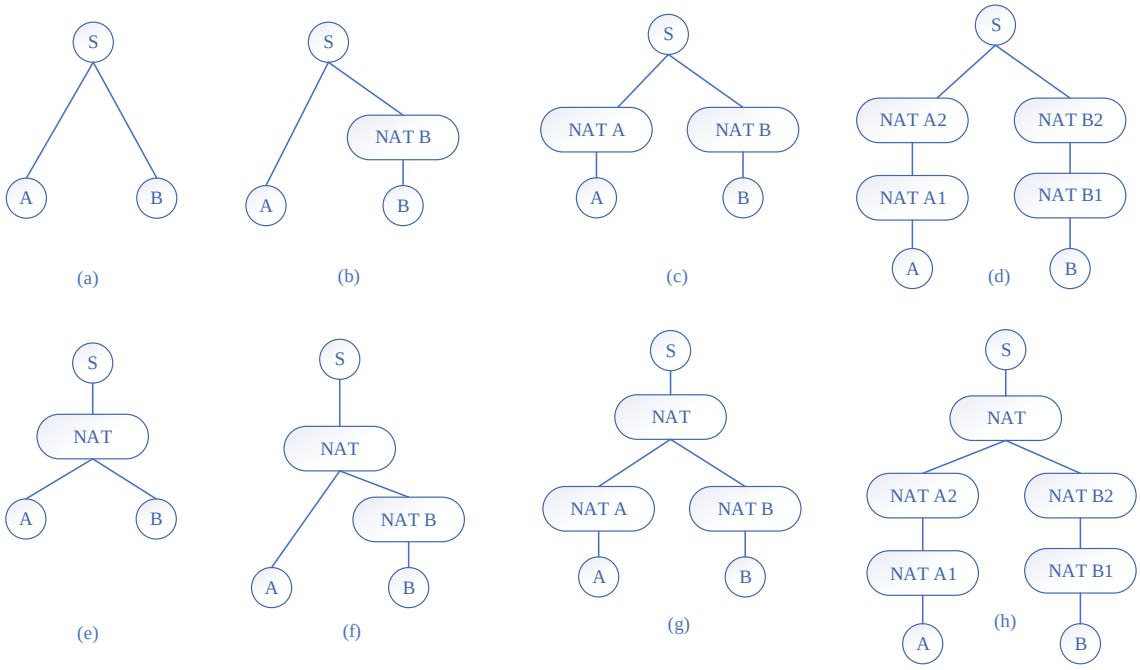

图 3.3 NAT 的分布情况[26]

我们可以将上述的 NAT 分布情况具体分为两大类：第一类，A、B 两台主机到服务器 S 的路线上不存在相交，即两台主机分属于不同的 NAT 之下(含多层 NAT)，属于这种类型的有(a)、(b)、(c)、(d)这四种情况；第二类，A、B 两台主机到服务器 S 的路线上存在相交，即两台主机在整体上属于同一个交点 NAT 设备，包括像(e)、(f)、(g)、(h)这四种情况。

本文的研究范围限定于第一种类型的 NAT 分布情况，第二种类型的 NAT 分布需要在第一种情况实现的基础之上，在处于交点的 NAT 设备之下设置一个额外的超级节点，由这个超级节点来代替实现服务器 S 的辅助穿越功能[26]。也就是说针对于第一种类型设计的 NAT 穿越方案可以直接应用于第二种类型，只不过不由服务器完成辅助穿越任务，而是通过额外的节点选取算法，选取一台距离 A、B 两台主机最近的交点 NAT 下的另一个直连节点作为超级节点完成辅助任务。

针对于上述的第一种类型的 NAT 分布情况，我们再根据 NAT 设备类型的不同进行更进一步的情况划分。在此之前有一点需要说明，即：在多层 NAT 进行串接时，可以将串接的多个 NAT 设备看作一个整体，整体 NAT 设备的类型取决于特性最严格的子 NAT 设备，按照特性严格程度的排序是：Symmetric NAT > Port Restricted Cone NAT > IP Restricted Cone NAT > Full Cone NAT[27]。因此，具体的穿越场景可以细分为以下 5 类。

1、场景一：双方皆为锥型 NAT

所谓锥型 NAT 指的是主机 A、B 可以属于 Full Cone 型、IP Restricted Cone 型、Port Restricted Cone 型这三种中的任意一种。在这种情况下，我们将使用一种名为 UDP Hole Punching(UDP 打洞)[13]的方案实现具体的 NAT 穿越。

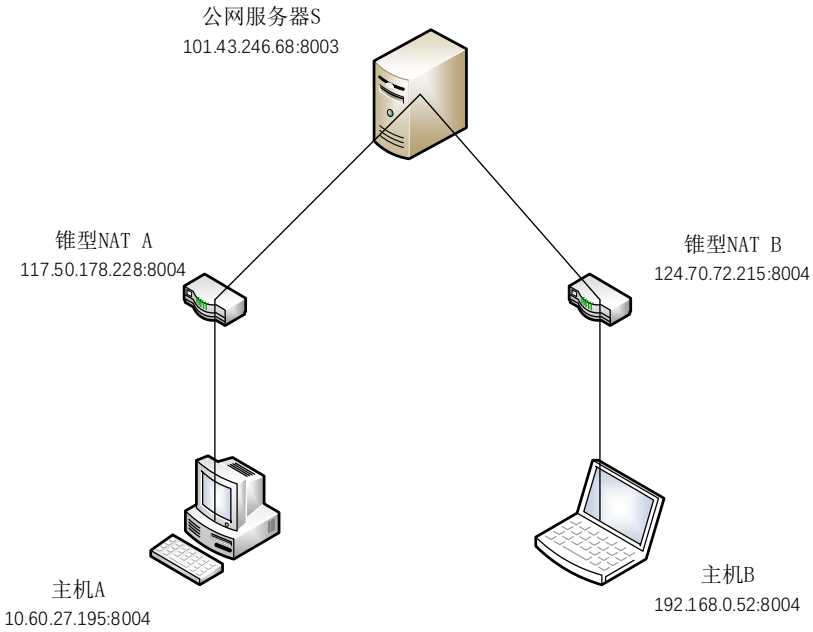

图 3.4 网络结构拓扑图 1

穿越步骤如下：

(1) 主机 A(10.60.27.195:8004) 和 B(192.168.0.52:8004) 分别向公网服务器 S(101.43.246.68:8003)发送一条注册消息，消息的内容一般可以是主机的用户名。服务器 S 借此可以分别获取此时通信主机 A 和主机 B 在 NAT 映射之后的公网地址信息，分别是 NAT A：(117.50.178.228:8004)和 NAT B：(124.70.72.215:8004)。

(2) 服务器 S 分别把对方的 NAT 映射后的网络地址进行告知，这样主机 A 获得的 B NAT 转换之后的地址(124.70.72.215:8004)，主机 B 获得了 A NAT 转换之后的地址(117.50.178.228:8004)。

(3) 假设主机 A 想要向主机 B 发送 UDP 数据包，首先它会向主机 B NAT 后的网络地址(124.70.72.215:8004)发送数据包，理所当然 NAT B 会拒绝这个数据包，因为 NAT B 上不存在与 NAT A 相关的映射记录，但是在 NAT A 上留下了与 NAT B 会话的记录，此映射关系记录为：(10.60.27.195:8004 - > 117.50.178.228:8004 - > 124.70.72.215:8004)。

(4) 主机 A 在向 NAT B 发送消息之后，紧接着向服务器 S 发送辅助请求消息，请求服务器 S 向主机 B 发送通知消息，通知主机 B 向 NAT A 的(117.50.178.228:8004)地址发送

消息，此时主机 B 也在自己的 NAT B 上留下映射记录(192.168.0.52:8004 －> 124.70.72.215:8004 -> 117.50.178.228:8004)。

(5) 来自于主机 B 的消息，当它到达 NAT A 时并不会被拒绝接收，而是顺利通过 NAT A 到达主机 A，理由是在 NAT A 上有(10.60.27.195:8004 - > 117.50.178.228:8004 - > 124.70.72.215:8004)的映射记录，因此 NAT A 将判断此消息是对自身在步骤(3)中发送消息的回复，允许其通过，并按照记录将其转发给主机 A。

(6) 同时，由于在 NAT B 上也留下了相关的映射记录，主机 A 也可以使用 10.60.27.195:8004 向主机 B 发送消息。至此，主机 A 和主机 B 可以进行通信，完成 NAT 穿越。

在上述的过程中，完成了主机 A 和 B 的跨 NAT 通信，但是需要注意的一点是：对于 UDP 通信来说，在 NAT 设备上留下的映射记录是有保留时间的，一旦通信双方长时间没有相互通信，相关的映射记录会被删除，那么下一次相互通信就需要再次由公网服务器进行辅助穿透。如果通信双方的两台主机 UDP 通信并不频繁，可以不考虑这一点，但如果通信比较频繁，为了防止每次打洞造成的通信延迟，有效的方式是在 NAT 穿越完成之后的两台主机间引入心跳包(Keep-Alive)机制，持续更新并保留 NAT 设备上的相关映射记录。

2、场景二：一方为非端口限制型的锥型 NAT,一方为对称型 NAT

此场景与场景一采用的 NAT 穿越方案相同，也是 UDP 打洞方案。但不同的是，由于一方存在对称型 NAT 设备，由于对称型 NAT 设备分别与服务器 S 和另一台主机通信时映射采用的网络地址是不同的，使得穿越变得困难，具体来说就是：只能由位于锥型 NAT 下的主机作为通信的首发起方，在自身的锥型 NAT 设备上留下映射记录，然后再由对称型 NAT 转发的消息利用这条映射记录穿越 NAT，最终完成通信。

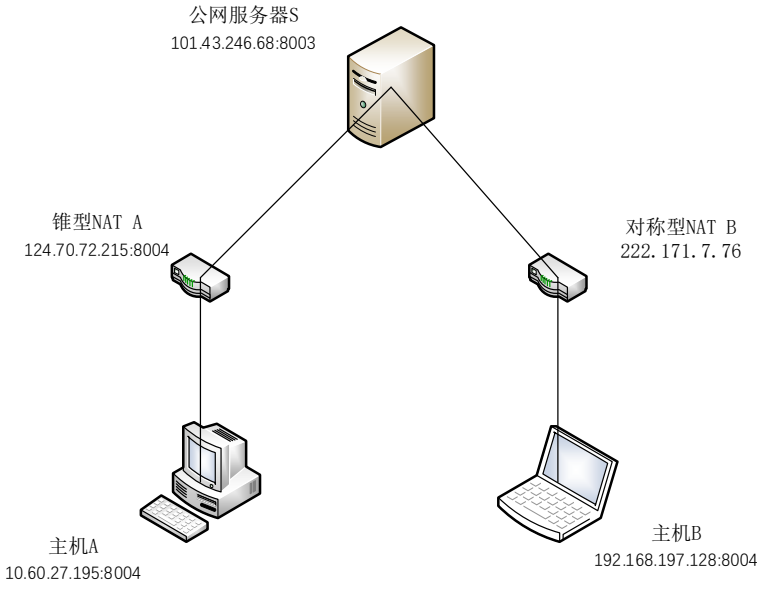

图 3.5 网络结构拓扑图 2

该场景下也同样采用 UDP Hole Punching 方案，因此步骤与场景一相同，不再赘述。UDP Hole Punching 方案只可以实现由锥型 NAT 下主机率先发送 UDP 消息情景下的 NAT 穿越，下面我们简单分析一下当对称型 NAT 下主机率先发送 UDP 消息时 NAT 穿越的可能性：

首先，步骤(1)、(2)是一样的；步骤(3)由主机 B 先向主机 A 发送数据，在 NAT B 上留下映射记录(192.168.197.128:8004 -> 222.171.7.76:8080 -> 124.70.72.215:8004)；步骤(4)中主机 A 在服务器驱使下向主机 B 发送消息，在 NAT A 上留下映射记录(10.60.27.195:8004 -> 124.70.72.215:8004 -> 222.171.7.76:8004)。但是，与上述方案不同的是，主机 A 发送的数据包不能通过 NAT B，而是会被丢弃，原因是主机 A 是将根据服务器处获取的 222.171.7.76 的 8004 端口作为目的地址的，但是主机 B 在 NAT B 上的映射地址是 8080 端口。至此，NAT 首次穿越过程失败。

此时，为了成功穿越，一种方式是在上次穿越失败之后，重新再进行一次穿越，此时由于 NAT A 上留有针对 IP 为 222.171.7.76 的“洞”，NAT B 发送的消息可以穿越 NAT A 到达主机 A，接着主机 A 再根据收到数据包的源地址 222.171.7.76:8080 进行回复就可以穿越 NAT B 再到达主机 B。但这种方案需要浪费一次“尝试进行 NAT 穿越的时间”，因为第一次穿越是注定失败的，第二次穿越必须等待第一次穿越失败后再次进行。另一种方案是交换双方的角色，一旦主机判断自身为对称型 NAT 且目的主机为锥型 NAT，此时主机就会尝试与对方主机“交换角色”，向对方主机发送一条请求对方向自己进行 UDP 打洞的消息，当然此消息需要由服务器 S 进行中转。

3、场景三：一方为端口限制锥型 NAT，一方为对称型 NAT

这里我们先按照与上述两场景相同的分析方法，详细研究此场景的 UDP 打洞方案的可行性。

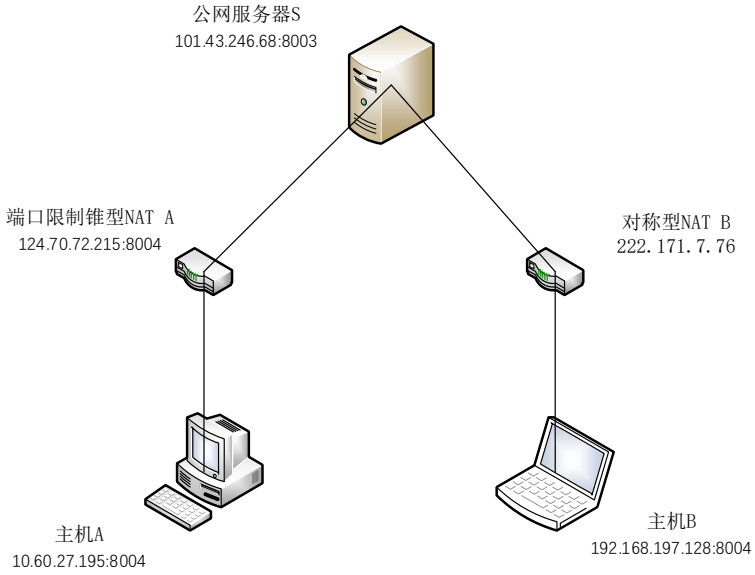

图 3.6 网络结构拓扑图 3

同样的，我们先以 NAT 类型相对比较简单的 NAT A 下的主机 A 作为通信的首发起方：步骤(1)、(2)、(3)、(4)相同，不再赘述；在步骤(5)中，不像场景二，由于 NAT A 对于外来数据包的 IP 地址和端口号的变化都是敏感的，因此从 NAT B 而来的数据包无法通过 NAT A，因此主机 A 也无法收到转发的消息，从而导致步骤(6)从根本上无法实现。至此，NAT 穿越失败，即使我们重新进行下一次 NAT 穿越也无法实现穿透。反过来，即使我们以主机 B 作为通信的首发起方，得到的结果也是一样的，无法实现 NAT 穿越。

因此，针对于场景三这种情况，单纯靠 UDP 打洞是没有办法实现 NAT 穿越的，我们必须采用其他技术来实现这个目的，也就是引入端口预测技术。下面将结合实际应用对端口预测技术进行说明：

首先通过分析我们可以知道：导致 NAT 穿越失败的原因发生在步骤(5)中，因为对称型的 NAT B 在前后两次映射中采用的端口号不同导致 NAT A 不允许其数据包通过。如果不考虑 NAT B 在具体映射时有何种规律性，我们可以这样考虑：既然失败的原因是 NAT A 上的端口映射记录其端口号与 NAT B 发来的数据包的端口号不匹配，那么只要我们在 NAT A 上多留下几条端口映射记录就可以解决这个问题，具体实现是让主机 A 在步骤（3）中用同一个端口向着 NAT B 的同 IP 下的若干个不同端口发送数据即

可，这样我们在 NAT A 上就留下了这样的多条映射记录：

（10.60.27.195:8004 - > 117.50.178.228:8004 - > 222.171.7.76:Port1）

（10.60.27.195:8004 - > 117.50.178.228:8004 - > 222.171.7.76:Port2）

（10.60.27.195:8004 - > 117.50.178.228:8004 - > 222.171.7.76:Port3）

……

上面的这些记录只有目的地址的端口号不一样。理论上看，由于任何网络的端口号都是属于 1~65535 这个范围内的，其中 1~1023 一般由系统保留，这样一来主机 A 只要向着 222.171.7.76 的 1024~65535 的所有端口都发送一条消息，当步骤（5）中 NAT B 的消息到达时必定可以被 NAT A 接收并转发给主机 A。

以上的方案虽然简单，但是我们一般不会采用，因为主机 A 必须要发送 64512 条消息，而且这些消息还需要在网络中转发，对网络带宽造成了大量无意义的占用。除了对带宽的消耗，假设我们的发包速率比较慢，仅为 100 packets/s，大约需要 10 分钟才能完成 NAT 穿越，这基本上是难以令人接受的速度。在实际应用时，我们将同时下面这种多端口+随机预测方式[5]：

首先，我们在步骤（3）中让主机 A 采用同一个端口向着 NAT B 的随机的 m 个端口发送数据，这样就在 NAT A 上留下 m 条映射记录。接着主机 B 在步骤 4 中采用 n 个随机的不同的端口发送数据给 NAT A 的唯一端口 8004。对于主机 A 的 m 次尝试，至少成功一次的概率为 P_{at_least1}，全部失败的概率为 P_{all_fail}，因此从概率上两者的关系为 $P_{at\_least1} = 1 - P_{all\_fail}$，因为主机 B 可能的端口个数为 $T = 65535 - 1024$，所以主机 A 这 m 次 NAT 穿越都失败的概率为 P_{all_fail}，如下式(3-1)。

$$P_{all\_fail} = \frac{T-n}{T} \times \frac{T-n-1}{T-1} \times \frac{T-n-2}{T-2} \times \frac{T-n-(m-1)}{T-(m-1)} = \prod_{j=0}^{m-1} \frac{T-n-j}{T-j} \tag{3-1}$$

其中，n 与 m 的取值范围皆为: $n,m \in [1,T]$。每次计算时保持 P_{all_fail} 不变，根据公式(3-2)就可以求出对应成功率下 n 与 m 的最优值。

$$\arg\underset{n,m}{ment}(n+m)\ s.t.\ \prod_{j=0}^{m-1} \frac{T-n-j}{T-j} = 1 - P_{at\_least1} = P_{all\_fail} \tag{3-2}$$

当 P_{all_fail} 为 0.01 时，即保证至少一次成功的概率为 99%时，最优的 n，m 分别取 521，563[5]。

4、场景四：双方皆为对称型 NAT

针对情景四这种情况，它比情景三更加更加严格。收发双方都位于对称型 NAT 之下，当主机 A 向 B 发送数据包，主机 A 的 NAT 后的公网地址改变，B 在服务器的通知

下向 A 发送数据时，B 的 NAT 后公网地址也发生改变。

如果我们采用与场景三一样的多端口+随机预测方式，此时假设有主机 B 绑定 256 个端口向 NAT A 发送消息，主机 A 用一个端口向 NAT B 的 2048 个端口发送消息，此时 NAT 成功穿越的概率是 0.01%[28]，原因很简单：原本主机 A 的 NAT A 的端口号只有一种可能性，而现在它有 2048 种可能性，对于主机 B 来说，它猜对 NAT A 端口号的概率为 1/2048。如果要想达到 99.9%的成功率，我们需要两边各进行 170,000 次探测 ——如果还是以 100 packets/s 的速度，就需要 28 分钟。相关数据表明一台 Juniper SRX 300 型号的路由器最多支持 64,000 active sessions。如果我们想创建一个成功的穿透连接，所创建的 NAT 映射记录基本是该路由器最大处理能力的 2.65 倍。显然，端口预测无法实现此场景下的 NAT 穿越，而在具体的应用场景下通常也是采用 TURN 中继方式来实现双对称型 NAT 下的穿透。

3.2.2 改进的端口预测方案

针对于上述的场景三和场景四，我们试图用多端口+随机预测方式对传统的 UDP Hole Punching 方案进行改进。其中，对于场景三我们最终达到的效果是：当(m,n) = (563,521)时，我们的 NAT 穿越可以实现高达 99%的成功概率。此时整个 NAT 穿越的实现需要发送 1084 个数据包，如果按最小计算，一个以太网帧需要占用 64 字节，总共大约需要 68KB，当然因为我们不可能不携带任何数据域部分，因此总体还会比 68KB 大一些。尽管从大小上看并没有消耗太多资源，但如果整个 P2P 网络中存在较多符合场景三的情况，就会造成比较大的开销浪费。针对这一问题，我们可以进一步对端口预测方案进行优化，为了实现优化就必须进一步对对称型 NAT 进行分类[5]，进一步缩小端口预测时的预测范围。

1、对 Symmetric NAT 的进一步分类

下面是对对称型 NAT 的进一步划分结果。

(1) 增量对称型 NAT

增量对称型第 i-1 次映射与第 i 次映射满足公式(3-1)，其中ΔP 是一个固定的常数，是增量对称型的端口增量。

$$Port_i = Port_{i-1} + \Delta p \tag{3-3}$$

(2) 恒等对称型 NAT

恒等对称型在 NAT 上端口没有被占用的时候，内部主机使用的端口和 NAT 映射转换的外部端口保持一致，但是一旦外部端口被占用了，其相应的映射端口就会往下递进。

这种 NAT 是最为常见的一种类型，因为它可以最大程度的利用有限的地址资源。缺点是一旦 NAT 下局域网规模比较大，映射端口递增的速度会很快，可能会表现为随机对称型 NAT。

(3) 全局随机对称型 NAT

该类型 NAT 每次都为内网主机对外访问的会话记录映射一个随机的端口号。

(4) 局部随机对称型 NAT

局部随机对称型在进行映射时虽然也是一个随机数，但每两次映射的端口号之间不会相差很远。

2、改进的端口预测方案

根据相关的映射特性，增量对称型也被称为可预测的对称型 NAT,全局和局部随机对称型被称为不可预测的对称型 NAT,而恒等对称型由于端口分配策略和本地端口相关，所以我们可以利用这个特性，将其变成可预测的对称 NAT 类型[5]。

对于两种不可预测的对称型 NAT，我们依旧还是使用多端口+随机预测的方式，但是对于可预测型的两种对称型 NAT，我们可以根据其映射特点，大幅度优化其端口预测策略。如果一方的对称型 NAT 设备属于上述的两种可预测型 NAT 中的任意一类，另一方为端口限制锥型 NAT，我们可以采用下述的改进方案。

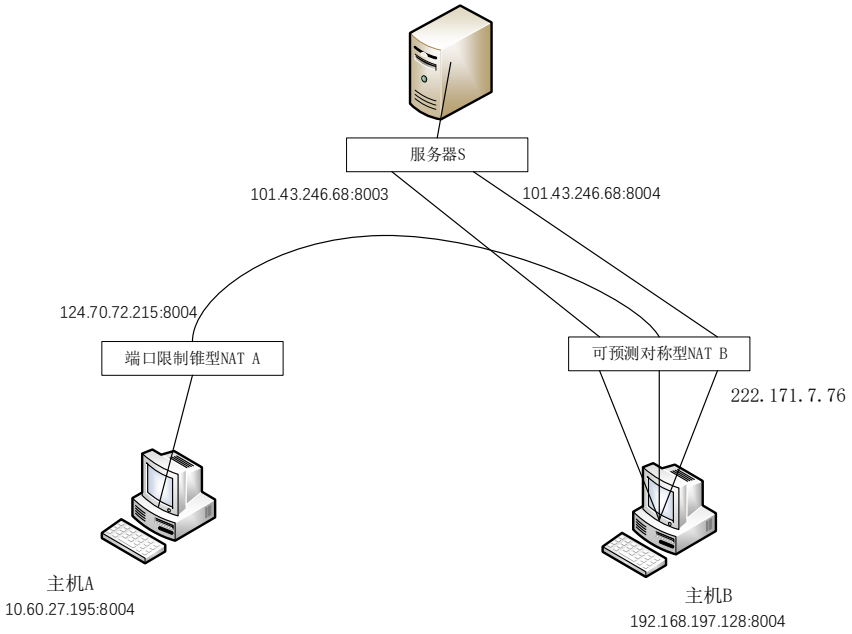

图 3.7 改进后端口预测方案网络拓扑图

具体的步骤如下：

(1) 主机 A(10.60.27.195:8004) 和 B(192.168.197.128:8004) 分别向公网服务器

S(101.43.246.68:8003)发送一条注册消息。服务器 S 借此可以分别获取此时通信主机 A 和主机 B 在 NAT 映射之后的公网地址信息，分别是 NAT A：(124.70.72.215:8004) 和 NAT B：(222.171.7.76:8004)。

(2) 服务器 S 分别把对方的 NAT 映射后的网络地址进行告知，这样主机 A 获得的 B NAT 之后的地址(222.171.7.76:8004)，主机 B 获得了 A NAT 之后的地址(124.70.72.215:8004)。由于 NAT B 每次在进行地址转换映射时端口号都会发生变化，因此服务器获取的 NAT B 的转换地址(222.171.7.76:8004)可以认为是无效的。但是，对于 NAT A 来说，其每次映射的端口号都是 8004，因此对于主机 B 来说，它获得的(124.70.72.215:8004)就是实际的主机 A 经过 NAT A 映射之后的地址。

(3) 接着，我们让主机 B 主动发送消息，而且是连续发送三条 UDP 消息。其中，第一条消息发送给服务器 S 地址(101.43.246.68:8003)，第二条消息发送给 NAT B 地址(124.70.72.215:8004)，第三天消息再次发送给服务器 S，但目的端口改为 8004。我们假设这三次发送中第一次消息在 NAT B 上映射的端口号是 Port_1，第二次消息映射的端口号是 Port_2，最后一次是 Port_3。根据可预测对称型的两种 NAT 的映射特性，Port_2 必然位于 Port_1 和 Port_3 之间，在 NAT B 上留下的映射记录为（192.168.197.128:8004 —> 222.171.7.76:Port_2 —> 124.70.72.215:8004）

(4) 对于服务器 S 来说，它可以明确地从接收数据包中获取到 Port_1 和 Port_3 的具体数值，然后服务器 S 将这两个数值作为实体数据发送给 NAT A 下的主机 A。主机 A 根据这两个作为上下边界的端口号就可以明确缩小对 Port_2 的预测范围(而且由于主机 B 是连续快速发送三条消息，因此 Port_3 ~ Port_1 是一个非常小的范围)。此时主机 A 就向着所有处于 Port_1~Port_3 之间的端口发送消息包，在 NAT A 上留下相应的映射记录并穿过 NAT B 到达主机 B。至此，我们完成了对可预测对称型 NAT 与端口限制锥型之间的穿透。

从理论上将改进前后的两种方案进行分析对比，根据表中数据可以看出，改进后的方案在性能上远优于原始方案。

表 3.3　改进前后的端口预测方案性能对比

	改进前的端口预测方案	改进后的端口预测方案
主动方发送消息数量	521 条	3 条
被动方发送消息数量	563 条	（Port_3 – Port_1）条
穿越成功率	99%	100%

针对于双方皆为对称 NAT 的场景，尽管由于可预测型 NAT 的特殊性，也可以对其端口预测方案做出优化改进，如文献[15]中介绍的 P-STUN 的 NAT 穿越方式，但其在具体使用时需要三台公网服务器同时进行辅助，且由于本方法没有考虑本机其他应用或者 NAT 下的其他主机访问外网也会引起端口增长的问题，导致其实用性不高，难以应用到实际开发，因此针对于双对称型 NAT 的场合依旧是采用 TURN 中继方式进行。

3.3 TCP 环境下的 NAT 穿越

3.3.1 TCP 穿越与 UDP 穿越的区别

TCP 协议与 UDP 协议最大的区别就是前者是一种面向连接的协议，面向连接的特性使得 TCP 相比于 UDP 具有了无边界收发、丢失重传、拥塞控制等多种优势。但在进行 NAT 穿越时，TCP 协议的这种面向连接特性反而使得其难以实现 NAT 穿越。

最重要的一点在于应用层程序接口，由于 TCP 连接套接字的 API 是为了实现构建 C/S 模式设计的[29]，它允许 connect()函数来主动建立从本机到目的终端的连接，也可以通过 listen()和 accept()函数等待外部主机发来的连接请求。但是这些函数全部是阻塞的，也就是说当客户端程序调用 connect()函数，除非连接成功或失败否则程序将一直阻塞在 connect()函数上，而且一个套接字(socket)最多同时调用一个 connect()函数进行 TCP 连接，这就导致了我们无法像 UDP 在端口预测中的那样一边向着某网络地址一次性发送大量数据包，一边又迅速转化为接收端等待对方的回复。即使我们采用多线(进)程方式加以改进，由于节点同时调用多个 connect()函数发送 SYN 都必须绑定不同的套接字 socket，这就导致锥型 NAT 的发送的每条 SYN 请求映射的也是不同的端口号，导致穿越难以实现。

3.3.2 针对非对称 NAT 场景的 TCP 穿越

以上的原因，导致了 TCP 场景下要实现 NAT 穿越是一件非常复杂又困难的事情，因为 TCP 各种 API 的阻塞特性，导致我们难以实现端口预测的功能。基于实验效果与当前对 TCP 场景下 NAT 穿越的研究现状，本方案仅针对非对称 NAT 场合下主机的 NAT 穿越进行设计探讨，针对于存在对称型 NAT 的场合，采取 TURN 中继的方式实现数据传输。

1、基于 TCP Hole Punching 的穿越方案

按照与 UDP Hole Punching 同样的思路，TCP Hole Punching 的 NAT 穿越方案也被提

出并应用于实践中，这种方案是一种实现难度最低也是使用最广泛的穿越方式，下面介绍该方案的具体实现步骤：

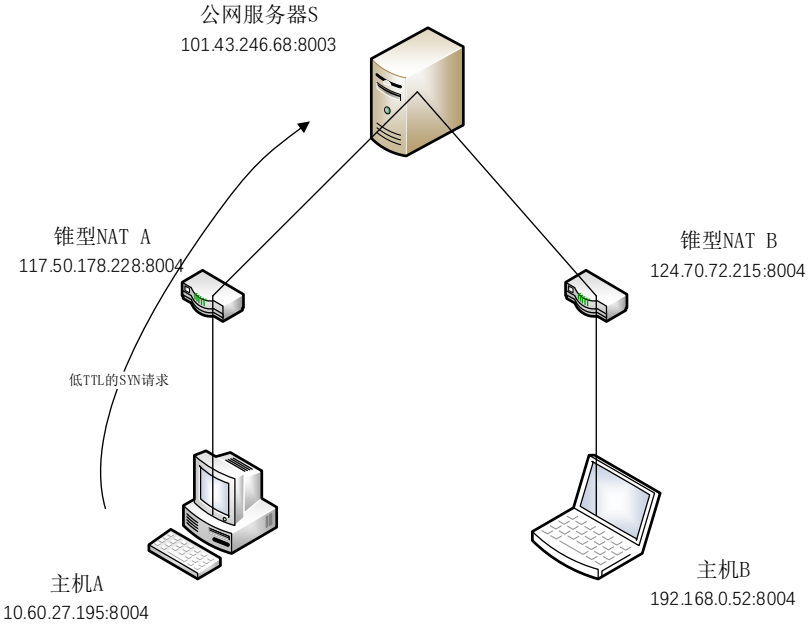

图 3.8　TCP Hole Punching 方案的网络拓扑图

(1) 主机 A(10.60.27.195:8004) 和 B(192.168.0.25:8004) 分别向公网服务器 S(101.43.246.68:8003)发送一条注册消息。服务器 S 借此可以分别获取此时通信主机 A 和主机 B 在 NAT 映射之后的公网地址信息，分别是 NAT A：(117.50.178.228:8004) 和 NAT B：(124.70.72.215:8004)。

(2) 服务器 S 分别把对方的 NAT 映射后的网络地址进行告知，这样主机 A 获得的 B NAT 之后的地址 (124.70.72.215:8004)，主机 B 获得了 A NAT 之后的地址 (117.50.178.228:8004)。

(3) 接着，假设由主机 A 作为 TCP 连接的首发方，它需要在自己的 NAT 设备上留下一条映射记录（10.60.27.195:8004 －＞117.50.178.228:8004 ->124.70.72.215:8004），可供后续过程中主机 B 发送的 SYN 连接请求能够通过。但与 UDP 打洞不同的是，由主机 A 发送的 SYN 包不能不加修改的直接发送至 NAT B，原因在于如果这样做了，NAT B 在收到这条陌生 TCP 连接的 SYN 包之后，不仅会将其丢弃，而且由于这条消息是被 NAT B 防火墙阻挡的，NAT B 还会向 NAT A 回复一条 RST 消息。当 NAT A 收到此 RST 回复消息后就会完全清除对此次 TCP 连接的“记忆”，也就删除了在 NAT A 上的相关映射记录，意味着这个由主机 A 在 NAT A 上打的“洞”被关闭了。为了解决这个问题，这条由主机 A 发送的 SYN 请求不能到达 NAT B，但是还需要穿

过 NAT A，这就要求我们必须控制 SYN 报文的 TTL，让其实现刚好穿过 NAT A 的效果，此时 NAT A 获得的不再是 RST 消息，而是由于超时产生的 ICMP 错误消息，而 ICMP 消息仅对部分 NAT 设备是比较敏感的，大部分 NAT 设备在收到 ICMP 错误消息时不会将 NAT 映射记录删除。

(4) 主机 A 发送 SYN 请求并在收到 ICMP 错误消息之后，选择对发送 SYN 所使用的端口进行 listen 监听，并且向服务器 S 发送辅助请求，请求服务器 S 通知主机 B 向自身发起 TCP 连接请求。

(5) 主机 B 收到服务器 S 的消息后，立即向 NAT A 的(117.50.178.228:8004)地址发送 SYN 请求，由于在 NAT A 上存在(10.60.27.195:8004 - > 117.50.178.228:8004 ->124.70.72.215:8004)的映射记录，因此这个 SYN 请求通过 NAT A 并被转发给主机 A，接着双方完成三次握手建立 TCP 连接。至此，主机 A 和 B 的 NAT 穿越完成。

这种方案原理上比较简单，唯一的难点在于主机 A 需要选择合适的 TTL 值在穿越自身 NAT A 且不到达 NAT B 的情况下，在 NAT A 上留下一条映射记录供后续主机 B 的 SYN 包通过。通常我们会考虑采用与 Traceroute 一样的思路：通过不断累加 TTL，并由沿途上的路由器进行回复就能得知穿越自身的最外层 NAT 设备抵达公网环境需要多大的 TTL 值。但是这个过程往往是比较耗费时间的。而且此方案还存在一个问题：发送的 TTL 必须确保刚好到达最外层 NAT 设备，也就是从主机 A 开始到第一个有公网 IP 的路由器，但通过使用 traceroute 测试可知，某些路由设备为了自身安全性(也可能是网络问题)，不会对 traceroute 探测消息回复超时消息，在 traceroute 显示时为“*”。这样我们就难以准确得知内网主机的最外层 NAT 的 IP 地址。可能导致 SYN 消息到抵达对方内网主机的最外层 NAT。从而导致 NAT 穿越失败。

2、基于 TCP Simulation Open 的穿越方案

TCP 打洞穿越 NAT 的方式比较简单，但是其穿越时耗费的时间很长，而且有机率失败。为了解决这个问题，我们采用一种基于 TCP 同时打开机制的新 NAT 穿越方案，下面是其穿越流程：

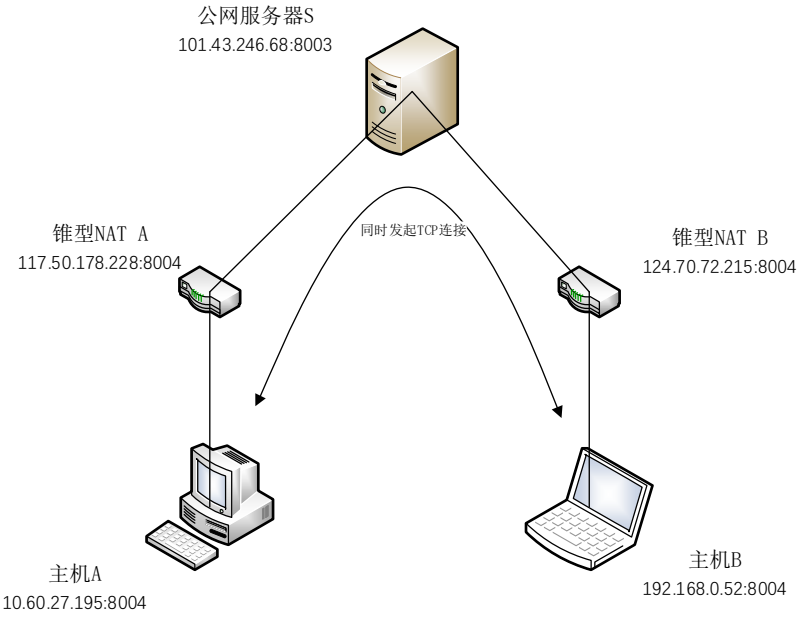

图 3.9　TCP Simulation Open 方案的网络拓扑图

步骤(1)、(2)不变，步骤(3)中主机 A 和主机 B 同时向对方映射的网络地址发起 TCP 连接请求。即主机 A 向(124.70.72.215:8004)发送 SYN 包，主机 B 向(117.50.178.228:8004)发送 SYN 包，这样双方的 SYN 包都在自己的 NAT 设备上留下一条映射记录后向对方的 NAT 设备发去，而在到达对方 NAT 设备之后又能从对方留下的映射记录中穿过 NAT 设备并转发到对方主机，进而通过 TCP 同时打开机制完成 TCP 连接。具体的 TCP 同时打开建立连接的过程如下：

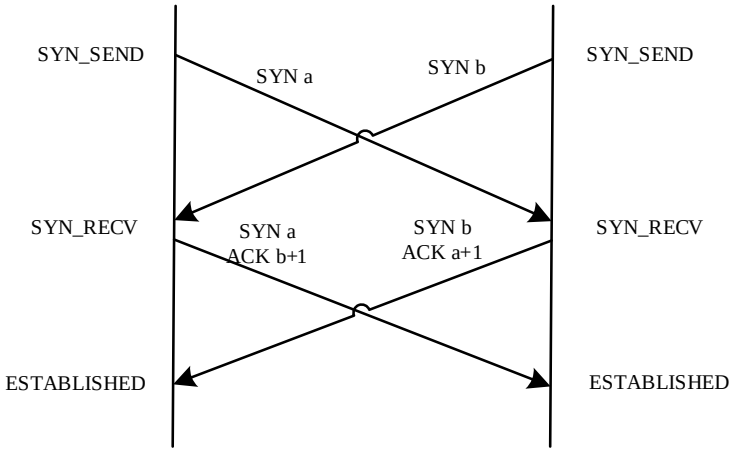

图 3.10　TCP 同时打开的连接时序图

(1) 主机 A 和 B 的 SYN 请求几乎同一时间到达对方主机(可以有时间差，但必须在 connect()因等待超时结束前抵达)。

(2) 两台主机在收到对方的 SYN 之后，分别向对方回复 SYN+ACK。

(3) 当双方再次收到对方的 SYN+ACK 之后就完成了 TCP 连接。

从上面的实现原理中我们可以看出，TCP Simulation Open 整体穿越过程相较于 TCP Hole Punching 更加简单，而且更加快速，基本上在调用 connect()函数的同时就能完成整个 TCP 连接。下面以表格的方式将这两种方案进行对比。

表 3.4 两种 TCP 场景下穿越非对称型 NAT 的方案对比

	时间耗费	流量耗费	成功率影响因素
TCP Hole Punching	TTL 探测+1 次服务器中转+1 次 TCP 连接	TTL 探测+服务器中转通知+TCP 的 3 次握手	低 TTL 包能否正确穿越最外层 NAT 设备
TCP Simulation Open	1 次服务器中转+1 次 TCP 完整连接	服务器中转通知+TCP 同时打开	双方主机 NAT 设备的层数是否差距不大

从上表中我们可以看出，TCP Simulation Open 方案在时间和流量耗费上都比 TCP Hole Punching 方案更加优越，但同时我们也注意到 TCP Simulation Open 方案实现的前提是双方主机发送的 SYN 包都能在对方 SYN 包到达自己 NAT 设备之前率先出去，在 NAT 设备上留下映射记录，否则对方 SYN 到达后会因为 NAT 不存在相关映射记录而被拒绝。

也就是说如果双方主机到达自身最外层 NAT 需要经过的 NAT 设备个数相差过大，有可能产生上述问题，导致穿越失败。但在实际的网络拓扑中，这种情况出现的不多，因为相比于穿越自身最外层 NAT 所需要的时间，从自身最外层 NAT 到达对方最外层 NAT 进行路由中转的时间消耗往往更多。因此，TCP Simulation Open 方案成功的概率在应用时仍然是比较高的，因此本设计在进行具体实现 TCP 穿越模块时采用了 TCP Simulation Open 方案。

3.4 本章小结

本章主要对论文所实现的 NAT 穿越方案的实现进行了理论介绍。首先介绍了传统的利用 STUN 协议进行 NAT 类型检测时的流程。然后针对 STUN 协议所规定的 NAT 类型检测流程进行分析，针对其消耗时间较长的缺点提出并采用了一种改进的 NAT 类型检测方案，并从理论上对两种方案的性能进行对比；接着是对 UDP 环境下 NAT 的穿越步骤进行介绍，根据不同的 NAT 场景分别介绍了标准的 UDP Hole Punching 方案和结合端口预测的 UDP Hole Punching 方案，而且将最为复杂的 Symmetric 型 NAT 进行进一步的类别划分，并对可预测型 Symmetric NAT 的端口预测方案进行了升级改进并从理论上介

绍其优越性；最后，针对在 TCP 环境下 NAT 的穿越方案进行介绍，由于 TCP 协议本身的复杂性，只针对非对称型 NAT 的穿越进行研究，并介绍两种可实用的穿越方案，并从理论上将这两种方案进行了性能对比，并选择 TCP Simulation Open 方案应用到本设计中。

第 4 章 系统设计

4.1 系统整体框架

本系统的总体设计目标是能够让位于不同 NAT 设备下的内网主机节点根据场景的不同，采取相对应的 NAT 穿越方案，借助于具有公网 IP 的主机节点（可以是混合式 P2P 的超级节点,又或者是集中目录式的中央服务器）加入到一个 P2P 网络中，而且能够实现在后续没有第三方服务器的场景下实现 UDP 或 TCP 通信。因此我们将整体系统分为客户端模块和服务器模块。

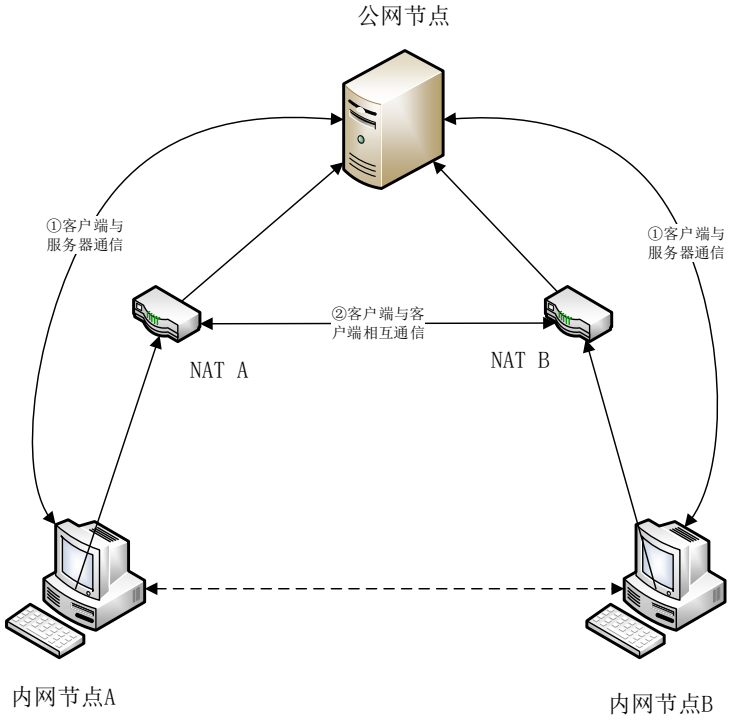

图 4.1 实现效果简图

每当一个节点想要加入 P2P 网络时，首先它需要向位于公网的节点发送登录注册消息。公网节点获得此登录请求后要判断该节点的 NAT 类型并将该节点的信息保留在本地，同时将目前网络中存在的其他节点的保留信息(包括用户名、公网上的 NAT 映射地址、NAT 类型等信息)发送给该新注册节点。

获取到 P2P 网络中其他节点的信息之后，当前节点根据需要可以选择与网络中的其他节点进行直接的数据通信，在公网节点的辅助下完成对双方 NAT 设备的穿透，建

立一条直连的数据链路(TCP 本身就是面向连接的，UDP 通过双方互发心跳包来维持一条数据链路)。

4.2 服务器模块的实现

服务器模块作为辅助两台位于不同 NAT 下搭载客户端模块的内网主机通信的桥梁，其本身需要运行在一台具有双公网 IP 的主机节点上，其主要的功能是对内网主机的 NAT 类型进行检测、辅助两台内网主机实现 UDP 通信或 TCP 通信，因此这里将整个服务器模块又分为三部分：NAT 类型辅助检测部分、UDP 辅助 NAT 穿越部分、TCP 辅助 NAT 穿越部分。

4.2.1 NAT 类型辅助检测部分

下面是服务器模块的 NAT 类型辅助检测部分的实现流程图：

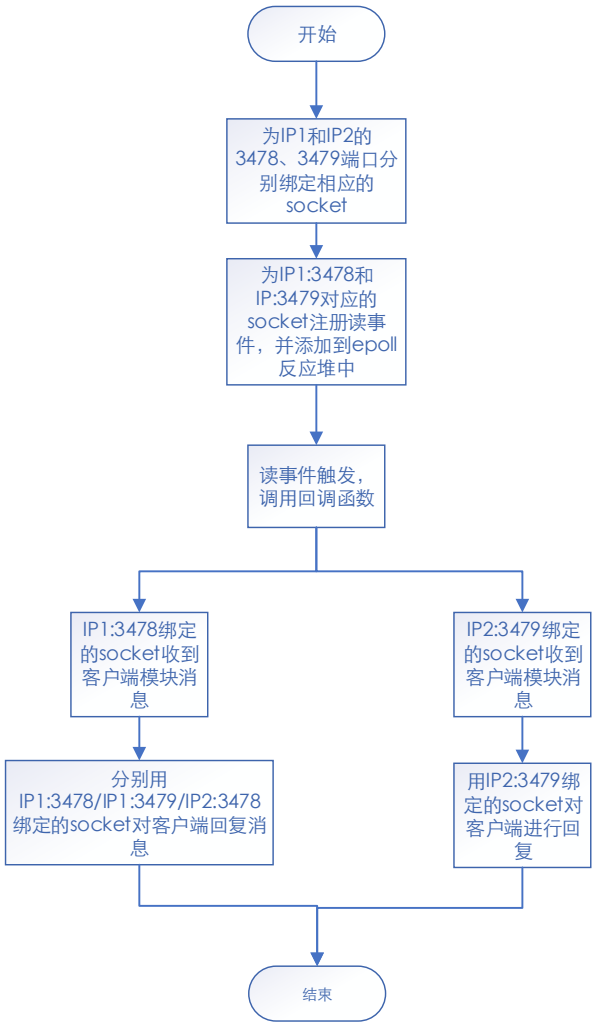

图 4.2 服务器模块 NAT 类型辅助检测部分流程图

当服务器模块启动运行时，它其中的 NAT 类型辅助检测部分整体以守护进程(后台进程)的方式运行，该部分将使用 IP1:3478/IP1:3479/IP2:3478/IP2:3479 四个地址二元组，并分别为这四个地址二元组绑定不同的 socket。由于只有 IP1:3478 和 IP2:3479 会接收到内网节点客户端模块的消息，因此此模块也只对这两个的 socket 进行监听。

由于服务器模块需要同时处理多个客户端的请求，这些请求有时是同时到达公网节点的，因此在程序设计时必须要考虑到并发问题，这里采用了 epoll 反应堆模型，将这两个进行监听的 socket 注册为读事件，为其设置读回调函数，并最终将其添加到 epoll 反应堆监听红黑树中，每当 socket 接收到数据时，相应的读事件被触发并被加入到处理队列中，epoll 反应堆将从前到后为处理队列中的触发事件依次自动调用其读回调函数。这里我们就是将公网节点向内网节点如何进行回复的程序部分放入到读回调函数中，每当公网节点收到内网节点的消息时就会自动调用回调函数，并根据接收到的消息类型判断该如何进行回复。如果是 IP1:3478 收到内网节点的请求，则用 IP1:3478/IP1:3479/IP2:3478 的三个 socket 分别对内网主机进行回复，消息的内容是内网主机 NAT 后的映射地址；若是 IP2:3479 收到消息，则仅用 IP2:3479 对内网节点进行回复，消息的内容同上。当处理完成之后，继续等待下一次读事件的触发。

4.2.2 UDP 辅助 NAT 穿越部分

这一部分负责辅助 P2P 网络中的两个内网主机之间实现 UDP 通信，与 NAT 类型辅助检测模块不一样，需要用一个新的端口进行监听，这里选择 IP1:8003 地址进行监听。本部分也作为守护进程运行在公网节点上。再具体介绍此模块之间，需要先介绍 UDP 通信过程中使用到的消息类型：

表 4.1 服务器模块 UDP 辅助 NAT 穿越部分消息类型

消息类型	意义	内容
LOGIN 消息	客户端节点登录消息	客户端节点用户名与 NAT 类型
LOGOUT 消息	客户端节点退出消息	无
P2P_TRANS 消息	客户端节点请求打洞协助	目标节点用户名
GET_ALL_USER 消息	客户端节点请求获取所有注册用户信息	无
P2P_SERVER_KEEP 消息	客户端节点发送的心跳包，用于维持连接	无

续表 4.1 服务器模块 UDP 辅助 NAT 穿越部分消息类型

消息类型	意义	内容
P2P_PORT_PROBE 消息	客户端请求端口扫描协助	目标客户端用户名/源客户端节点 NAT 映射后的两个端口号
P2P_CHANGE 消息	客户端请求让目标节点向自身发起 UDP 打洞	目标节点用户名
P2P_RELAY_ACK 消息	目的客户端节点对中转消息的回复	源节点 NAT 转换后的 IP:PORT 地址
P2P_USER_INF 消息	公网节点对 GET_ALL_USER 和 LOGIN 消息的回复	公网节点保存的其他登录节点的信息
P2P_SOMEONE_WANT_TO_CALL_YOU 消息	公网节点收到 P2P_TRANS 消息后，向目标节点发送的消息	源客户端节点 NAT 映射后的 IP:PORT 二元组

公网节点循环监听 IP1:8003 绑定的 socket，每当有内网主机消息到达时，根据消息的类型进行不同的数据处理操作：

1、 LOGIN 消息：当公网节点收到 LOGIN 消息时，首先将查询本地数据库中是否已存在该用户，若不存在将其用户名、NAT 映射后的 IP:PORT 存入本地数据库。最后，将本地数据库中所有的用户信息发送给该内网节点(P2P_USER_INF 消息)，由其进行存储。

2、 LOGOUT 消息：当公网节点收到 LOGOUT 消息时，首先将查询本地数据库中是否已存在该用户，若已存在则删除该用户的所有注册信息。

3、 GET_ALL_USER 消息：内网用户每次需要与其他内网主机进行 NAT 穿透之前，需要向公网节点发送此消息重新更新本地保存的其他用户信息，防止目标用户已退出 P2P 网络。

4、 P2P_TRANS 消息：当公网节点收到此消息后，根据消息中的目标节点用户名在数据库中查找对应的 NAT 映射后的 IP:PORT 地址。然后向该地址发送 P2P_SOMEONE_WANT_TO_CALL_YOU 消息，内容为源主机节点的 NAT 后的网络地址

5、 P2P_SERVER_KEEP 消息：对于某些恒等对称型 NAT 来说，其每次与公网节点进行 UDP 使用的端口号不一样，这会导致服务器可能有时无法主动向这种内网节点发送消息，这里选择由内网节点每 4s 向公网节点发送心跳包，一旦端口号变更公网节点能获取到变化之后的端口并在本地数据库中进行更新。

6、 P2P_PORT_PROBE 消息：公网节点临时设置一个新的 socket 绑定新的端口号，公网

节点将收到两条来自于同一内网节点的此消息(8003 端口 socket 和新的 socket 分别负责接收一条消息)，这两条消息 NAT 映射后的 IP:PORT 地址不同，分别为 IP:PORT1 和 IP:PORT2，但两条消息的内容相同，都为目标节点的用户名。公网节点将这两个端口号 PORT1 和 PORT2 作为消息的内容发送给目标节点，消息类型为 P2P_SOMEONE_WANT_TO_CALL_YOU。

7、 P2P_CHANGE 消息：对于一方为可预测对称型 NAT，一方为端口限制锥型 NAT 的场合下。必须由可预测对称型 NAT 下的内网主机作为数据的首发方，然后由端口限制锥型 NAT 进行预测，因此如果是端口限制锥型 NAT 下主机作为首发方，必须先交换双方角色，然后在执行 NAT 穿越。

8、 P2P_RELAY 消息：针对于无法实现 NAT 穿透的场景，内网主机将选择发送此类消息携带信息发送给公网节点，由此公网节点负责将此消息转发给目标节点

9、 P2P_RELAY_ACK 消息：当目标节点收到此消息后，会向公网节点回复此消息，再由公网节点根据内容将其转发回源节点，告知其转发已结束。

4.2.3 TCP 辅助 NAT 穿越部分

这是服务器模块的最后一部分，目的是协助两台内网主机实现 TCP 通信。此部分监听的端口号是 7172，并作为守护进程运行在公网节点的后台。同样的，介绍在此部分中公网节点使用的消息类型：

表 4.2 服务器模块 TCP 辅助 NAT 穿越部分消息类型

消息类型	意义	内容
P2P_TRANSFER_PING 消息	客户端节点以 TCP 模式登陆到公网节点的登录消息	客户端节点用户名和 NAT 类型
P2P_TRANSFER _GET_ALL_USER_LIST 消息	客户端节点申请从公网节点获取其他节点信息	无
P2P_SERVER_USERINF 消息	公网节点对前两条消息的回复	当前公网节点保存的所有注册节点的信息
P2P_TRANSFER_QUERY_ DEVICE_INFO_REQUEST 消息	源客户端节点请求公网节点辅助连接的请求消息	目的客户端节点的用户名
P2P_TRANSFER_QUERY_ DEVICE_INFO_RESPONSE 消息	公网节点向源客户端节点的回复	无

续表 4.2 服务器模块 TCP 辅助 NAT 穿越部分消息类型

消息类型	意义	内容
P2P_TRANSFER_PUNCH_HOLE 消息	公网节点向目的客户端节点发送的消息	源客户端节点 NAT 后的 IP:PORT 地址

以上便是公网节点的 TCP 辅助 NAT 穿越模块会使用到的所有消息类型。整个模块采用了 libevent 轻量级网络库进行实现，支持高并发场景。Libevent 网络库是基于事件驱动的，所有使用到的 socket 都绑定一个 bufferevent 事件并注册其回调函数，每当 socket 上的事件被触发时就会自动调用其注册的回调函数，因此具体的操作都由回调函数处理解决。下面介绍程序中使用到的几个重点的 bufferevent 与其回调函数的功能：

1、 服务器连接监听器 listener：绑定的 socket 正是等待客户端节点 TCP 连接的那一个，回调函数是 listener_cb()，负责接受来自于客户端节点的 TCP 连接请求并负责创建对应的连接 socket，最后为此连接 socket 绑定对应的 bufferevent 和读回调函数 conn_read_cb()以及事件回调函数 conn_event_cb()。

2、 连接 socket 对应的 bev：上述创建的与客户端节点 connect_sock 对应的 bufferevent 事件，每当客户端节点通过 TCP 连接向公网节点传输数据或断开连接时都会触发此 bufferevent。对应的回调函数由两个：conn_read_cb()和 conn_event_cb()，前者负责处理数据传输相关问题，后者负责处理连接断开相关问题。conn_event_cb()当与客户端节点的 TCP 连接出现异常时会被调用，将在本地数据库中删除该客户端节点对应的信息，释放其有关的资源。conn_read_cb()会根据接收到消息的消息类型采取下述不同的措施：

(1)消息类型为 P2P_TRANSFER_PING：在本地数据库中搜索消息中指定用户的用户名是否存在，若不存在则在本地保存该客户端节点的注册信息。最后向该客户端节点回复 P2P_SERVER_USERINF 消息。

(2)消息类型为 P2P_TRANSFER_GET_ALL_USER_LIST：向该客户端节点回复 P2P_SERVER_USERINF 消息。

(3)消息类型为 P2P_TRANSFER_QUERY_DEVICE_INFO_REQUEST：提取消息中的目标节点用户名并在本地数据库中匹配查询对应的网络地址。准备好两条消息，一条消息为 P2P_TRANSFER_QUERY_DEVICE_INFO_RESPONSE 类型，不含任何有效内容只负责通知源主机节点；另一条消息为 P2P_TRANSFER_PUNCH_HOLE 类型，内容为源主机节点的 NAT 后的网络地址和用户名。公网节点将在同一时刻同时向着源主机节点和目的主机节点分别发送这两条消息。

4.3 客户端模块的实现

客户端模块运行于所有的希望或已加入 P2P 网络的主机上，负责探查主机所属的 NAT 设备的类型，并根据自身与对方 NAT 类型的组合场景选择合适的 NAT 穿越方案，在运行服务器模块的公网节点的帮助下完成 UDP 或是 TCP 环境下的跨 NAT 设备的主机 P2P 通信。这里将整个客户端模块也分为三部分：NAT 类型检测部分、UDP 下 NAT 穿越部分、TCP 下 NAT 穿越部分。

4.3.1 NAT 类型检测部分

下面是客户端模块的 NAT 类型检测部分的流程图：

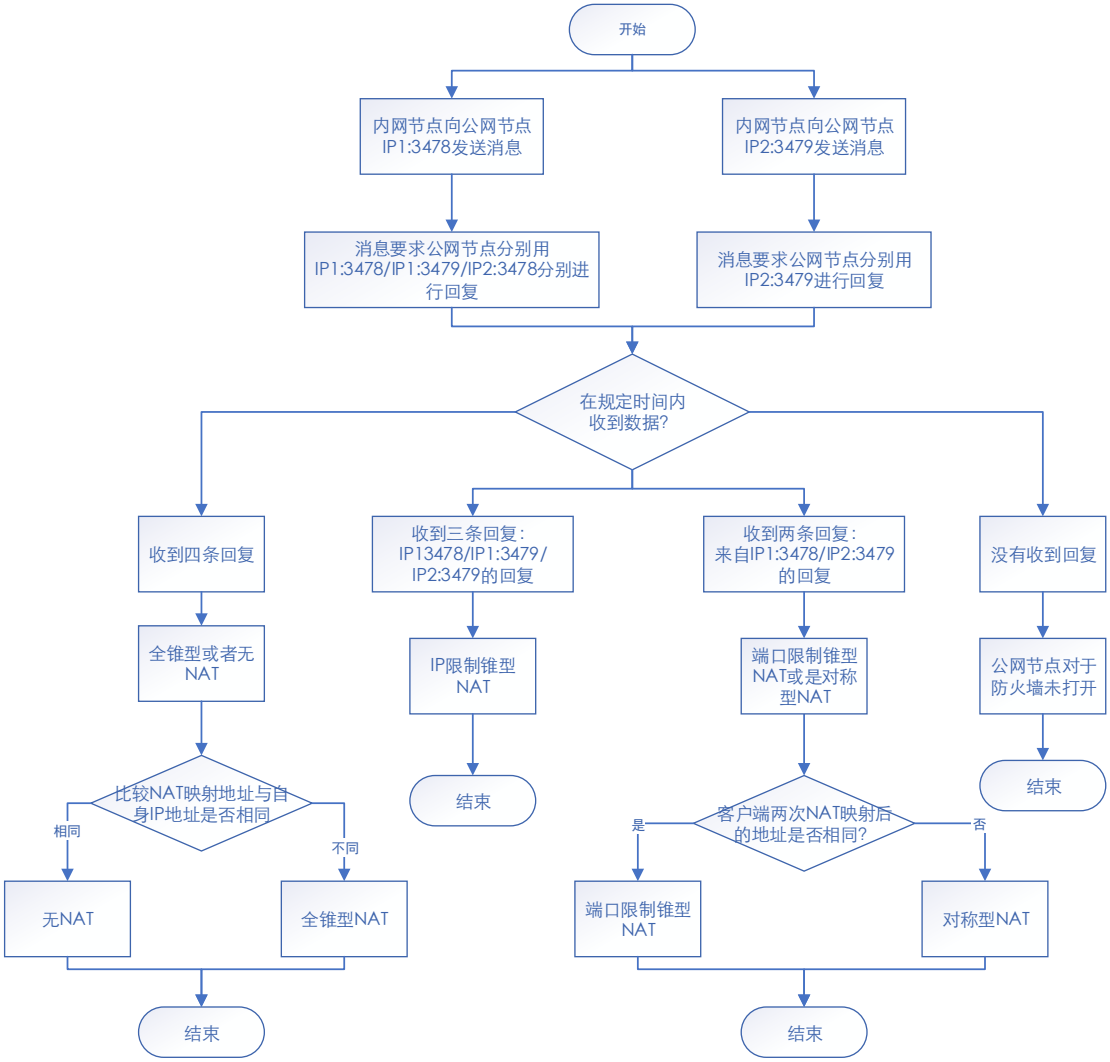

图 4.3 客户端模块 NAT 类型检测部分流程图

NAT 类型检测部分负责向公网节点不同地址发送探测包，根据公网节点回复包的数量和内容确定自身的 NAT 类型：首先，内网节点分别向公网节点的 IP1:3478 和 IP2:3479

发送 UDP 探测包，由于公网节点在进行回复时是同时发送多个回应包的，因此内网节点也需要接收的并发问题。这里我们采用 epoll 模型负责监听内网主机发送消息的 socket，将该 socket 接收到的消息加入到消息队列中，依次进行处理，根据该 socket 最终接收到回复包的数目对 NAT 进行粗略的分类判断，然后再根据回复包的内容对 NAT 类型进行具体判断。在完成判断之后，结束该部分并将检测得到的 NAT 类型传给下一模块。

4.3.2 UDP 下 NAT 穿越部分

在获取自身的 NAT 类型之后，内网主机可以选择与其他网络中的节点进行 UDP 通信。本部分负责具体实现 UDP 场景下 NAT 穿越，整个程序分为三个线程：主线程，负责数据的发送，包括像公网节点发送辅助请求以及向其他内网节点发送 UDP 数据包；接收子线程，负责接收来自于公网节点和其他内网节点发送而来的数据，根据数据类型的不同采取不同的措施；心跳包子线程，负责向公网节点和成功完成 NAT 穿越的其他内网节点进行有选择的心跳包发送，所谓有选择就是对于长时间没有进行 UDP 通信的内网节点不在发送心跳包。下面针对这三个线程进行具体的介绍：

1、主线程

请求用户输入自身用户名，然后向公网节点发送 LOGIN 消息并等待其回复的 P2P_USER_INF 消息，提取该消息中内容，将所有其他节点的注册信息保留在本地。之后进入循环，循环的内容是输出用户菜单提示用户进行相关操作，用户可输入的指令分为三类：Exit 指令用于退出 P2P 网络，向公网节点发送 LOGOUT 消息；Getu 指令用于向公网节点重新获取活跃节点的注册信息，向公网节点发送 GET_ALL_USER 消息；Send 指令用于向指定用户名的对方内网节点发送消息，具体格式为 Send [用户名] [文本消息]。

其中，Send 指令会根据从公网节点获取到的对方内网主机的 NAT 类型与自身 NAT 类型组合场景的不同，选择不同的 NAT 穿越方案：

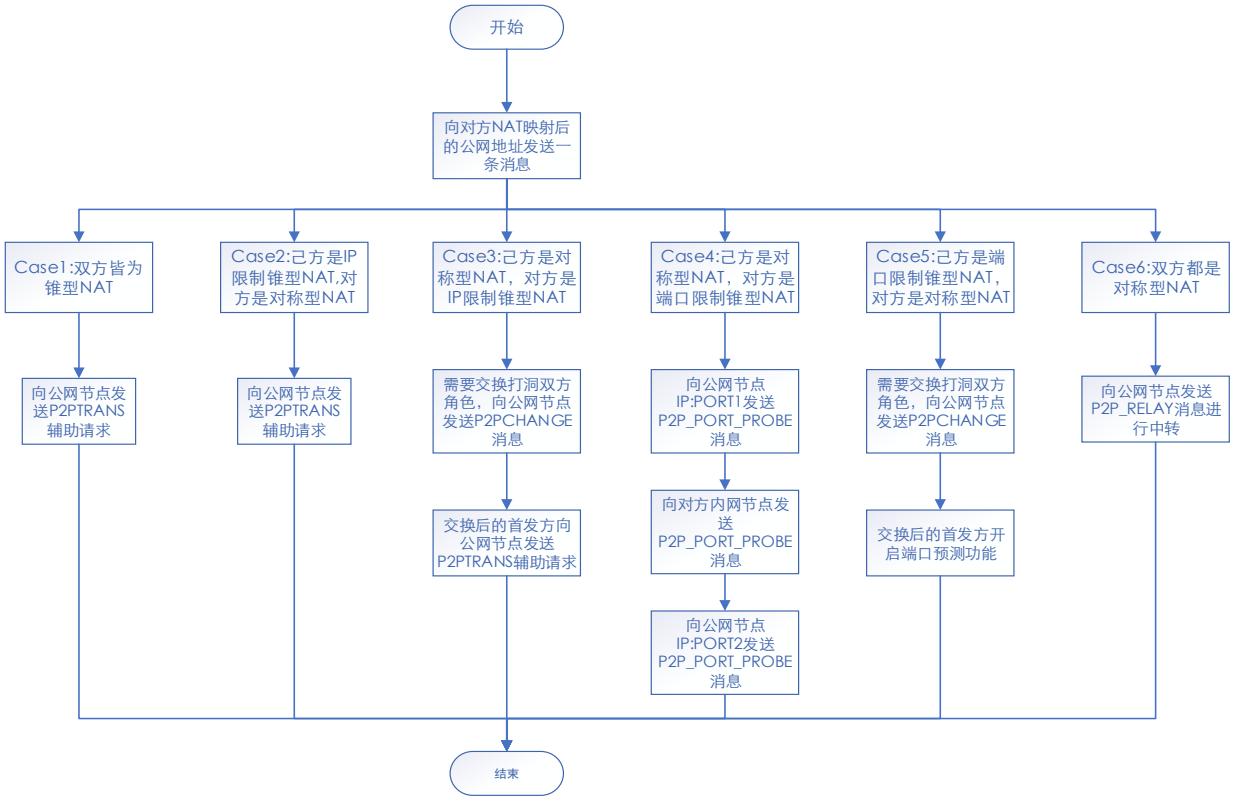

图 4.4 客户端模块 UDP 下 NAT 穿越部分主线程流程图

针对于情况 3 的说明：对于情况 3，在理论分析阶段我们指出这种场景下一次 UDP Hole Punching 方案无法实现 NAT 穿越，因此有必要进行第二次。但是此方法时间效率比较低，需要在已知第一次穿越必定失败的前提下等待第二次穿越。因此这里我们也采用 P2P_CHANG 消息，交换双方角色，实现一次 UDP Hole Punching 即可成功的方案。

2、接收子线程

接收子线程负责接收所有其他节点发送而来的消息，并根据消息的不同，采取不同的措施，再次之前需要介绍两种专门用于内网节点之间传输的消息类型：

表 4.3 UDP 下专用于内网节点之间传输的消息类型

消息类型	意义	内容
P2P_MESSAGE	向对方内网节点发送的文本消息	文本消息
P2P_MESSAGE_ACK	对方内网节点对 P2P_MESSAGE 的回复	无

下面介绍接收子线程在收到各类消息后采取的措施：

(1) P2P_MESSAGE 消息：显示对方主机 NAT 后的公网地址以及文本消息的内容，根据

公网地址向对方回复 P2P_MESSAGE_ACK 消息。

(2) P2P_MESSAGE_ACK 消息：显示对方主机 NAT 后的公网地址。更新与对方主机的最后一次通信的时间，便于向对方发送心跳包。

(3) P2P_SOMEONE_WANT_TO_CALL_YOU 消息：此消息的来源必定是公网节点，提取消息中源内网主机 NAT 后的公网地址，向其发送 P2P_TRASH 包。

(4) P2P_TRASH 消息：显示对方主机 NAT 后的公网地址。一旦接收到此消息，说明双方的 NAT 穿越已完成，需要向对方重新发送 P2P_MESSAGE 消息。

(5) P2P_USER_INF 消息：此消息必定来自于公网节点，提取消息中的所有其他主机的注册信息，并对本地数据库进行更新。最终在终端显示所有用户的用户名、NAT 后的 IP:PORT、NAT 类型等信息。

(6) P2P_CHANGE 消息：此消息必定来自于公网节点，提取消息中源内网主机的用户名并在本地查找。若查找到对于用户，判断自身 NAT 类型，若为 IP 限制锥型则采取 UDP Hole Punching 方案进行 NAT 穿越；若为对称型 NAT 则采取端口预测+UDP Hole Punching 方案进行 NAT 穿越。

(7) P2P_PORT_PROBE 消息:此消息必定来自于公网节点，提取消息中源客户端节点 NAT 映射后的两个端口号，进行端口预测，向着预测的源内网节点地址发送 P2P_TRASH 消息。

(8) P2P_KEEP 消息：收到了对方内网节点的心跳包，向对方回复 P2P_KEEP_ACK 消息。

(9) P2P_KEEP_ACK 消息：收到了对方对心跳包的回复，更新与对方主机最后一次通信的时间。

(10) P2P_RELAY 消息：此消息必定来自于公网节点，提取消息中的源端内网节点的 IP:PORT 以及中转的文本消息，进行显示。最后向公网节点回复 P2P_RELAY_ACK 消息，内容是源端主机的用户名。

(11) P2P_RELAY_ACK 消息：此消息必定来自于公网节点，一旦受到此消息，说明消息的中转已完成。

3、心跳包子线程

此线程负责向实现了 UDP 通信的其他节点发送心跳包。心跳包的发送频率是每 1s 一次，发送的对象是在 4s 以内曾与本机通信过的内网节点。具体的我们在主机收到 P2P_MESSAGE_ACK 和 P2P_KEEP_ACK 消息中对主机双方的通信时间进行了更新。这样做的目的是防止向着 P2P 网络中某些已退出的内网节点发送心跳包。

4.3.3 TCP 下 NAT 穿越部分

在获取自身的 NAT 类型之后，内网主机可以选择与其他网络中的节点进行 TCP 通信。该模块同样采用 libevent 网络库实现。

首先，客户端节点需要与公网节点发送 TCP 连接请求，成功之后为连接 socket 绑定对应的 bufferevent 事件以及相应的回调函数，也包括了读回调函数 conn_read_cb()与事件回调函数 conn_event_cb()，分别在收到公网节点数据和与公网节点 TCP 连接出现问题时被调用。接着，客户端节点将主动向公网节点发送 P2P_TRANSFER_PING 消息，消息的内容是本节点的用户名和 NAT 类型。

客户端节点第一次收到的消息将是来自于公网节点对于 P2P_TRANSFER_PING 消息的回复：P2P_SERVER_USERINF 消息，根据对接收消息类型的判断，客户端节点将此消息中的其他节点登记信息进行保存，同时创建一条写线程专门负责与公网节点或其他内网节点发送消息或连接请求，对应的线程函数为 command_send_thread()，至此本回调函数结束。

接着，客户端节点将进入写线程的线程函数 command_send_thread()。该线程函数的运行流程如下：

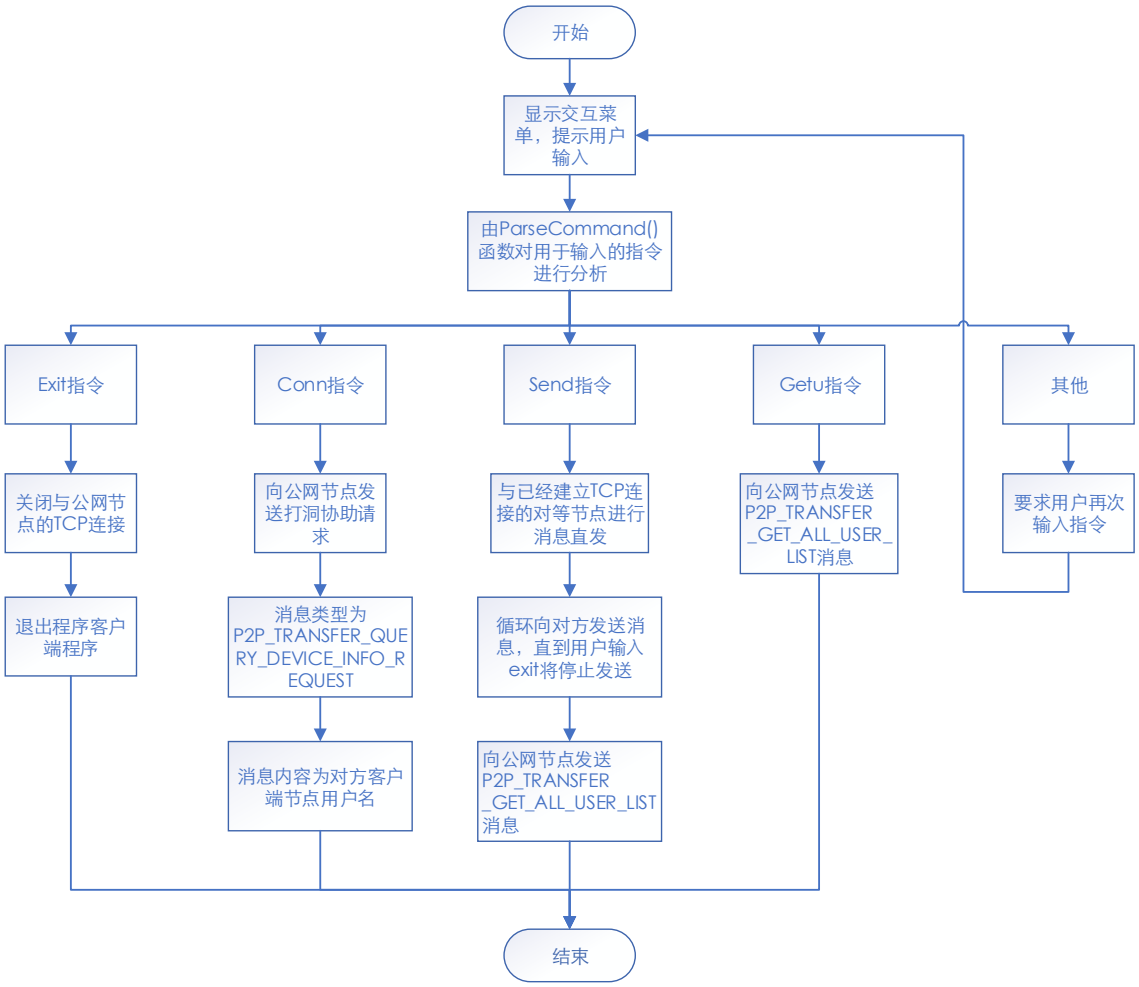

图 4.5 客户端模块 TCP 下 NAT 穿越部分写线程流程图

当客户端节点作为 TCP 连接的首发起方，输入 Conn 指令向公网节点发送 P2P_TRANSFER_QUERY_DEVICE_INFO_REQUEST 消息时，公网节点会同时并分别向源内网节点和目标内网节点发送 P2P_TRANSFER_QUERY_DEVICE_INFO_RESPONSE 消息和 P2P_TRANSFER_PUNCH_HOLE 消息。

当客户端节点收到 P2P_TRANSFER_QUERY_DEVICE_INFO_RESPONSE 消息时，它将调用 connect()函数不断向目标内网主机发送 TCP 连接请求(为了提高连接成功率，改为连续十次发送)。一旦 connect()连接成功就返回对应的连接 socket，并为该连接 socket 绑定 bufferevent 事件和读回调 direct_conn_read_cb()以及事件回调 direct_conn_event_cb()，分别负责处理来自于目的主机发送来的消息和 TCP 连接异常事件，同时我们将此连接 socket 以及 bufferevent 作为 value，对方主机的用户名作为 key，存放到本地数据库中。最后，再次创建一条写线程，用于向该目的主机进行数据发送。

当客户端节点收到 P2P_TRANSFER_PUNCH_HOLE 消息时，将采取与上述相同的

步骤，唯一的区别在于因为处于消息接收方的身份，因此不需要额外创建写线程。

4.4 本章小结

本章主要介绍了整个 NAT 穿越方案的编程实现思路，包括整个系统框架以及具体模块的设计实现，介绍了使用到的各类消息的应用场景、内容以及每个模块的运行机制，并给出了相关的程序流程图。

第 5 章 系统测试

5.1 测试环境

本文的测试环境如下图所示，测试服务器 S 作为公网节点有两个公网 IP 地址，两台客户端节点分别位于不同的 NAT 设备之下。服务器与内网主机均为 Linux 系统。

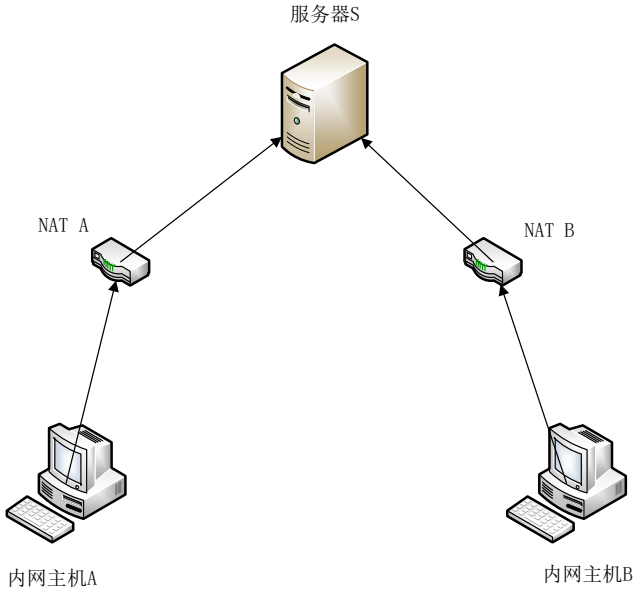

图 5.1 测试环境拓扑图

测试过程中分别使用虚拟机(校园网)、Ucloud 云主机(EIP)、腾讯云主机(EIP)、华为云主机(EIP)分别作为内网主机进行组合测试(校园网路由器和 EIP 路由器作为其各自的 NAT 设备)。

5.2 测试结果与分析

1、公网服务器 S 需要分别在后台运行 NAT 类型辅助检测模块、UDP 辅助穿越模块、TCP 辅助穿越模块。(为了显示出测试结果，这里并没有转入后台，而是单独对每一模块进行测试显示)

```
ubuntu@VM-16-15-ubuntu:/NatTypeNew/nattype-new$ ./server
call bind(), sock: 3, ip: 172.16.16.15, port: 5000
call bind(), sock: 4, ip: 172.16.16.15, port: 5001
call bind(), sock: 5, ip: 172.16.0.5, port: 5000
call bind(), sock: 6, ip: 172.16.0.5, port: 5001
enter event_dispatch()...
```

图 5.2 服务器开启 NAT 类型辅助检测模块

NAT 类型检测模块中，并没有选择标准的 STUN 的 3478/3479 端口，而是选择了 5000/5001 端口，但这并不影响方案的效果。

2、内网主机 A 和 B 上的客户端需要调用 NAT 类型检测模块，对自身 NAT 类型进行探测。同时主机 A 和 B 需要选择采用 UDP 通信还是 TCP 通信，并将自身信息向服务器 S 注册。

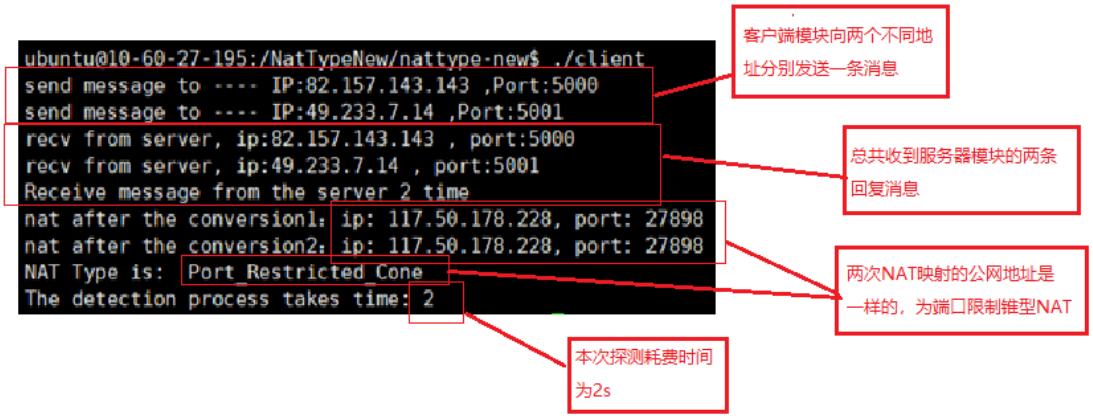

图 5.3(a) 端口限制锥型 NAT 下主机的 NAT 类型测试结果

图 5.3(b) 对应场景服务器辅助测试结果

可以看到，该内网主机在进行 NAT 类型探测时，分别向公网服务器 S 的 82.157.143.143:5000 和 49.233.7.14:5001 发送了数据包。其最终从服务器 S 处收到了 2 条回复，两条回复分别来自于服务器 S 的 82.157.143.143:5000 和 49.233.7.14:5001 地址。因为两次 NAT 映射后的公网地址是相同的，因此 NAT 类型为端口限制锥型。整个过程耗时 2

s。

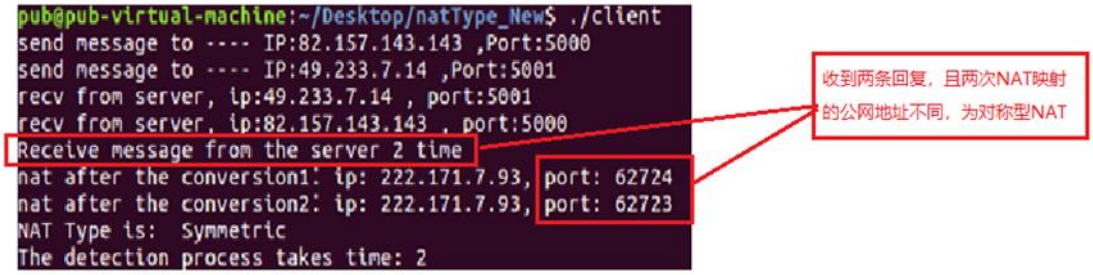

图 5.4(a) 对称型 NAT 下主机的 NAT 类型测试结果

```
read event happend...
recv message from client ,ip:222.171.7.93,port:62724
message Type :3
server ip :172.16.0.5,  port :5001 send to ---> client

read event happend...
recv message from client ,ip:222.171.7.93,port:62723
message Type :1
server ip :172.16.0.5,  port :5000 send to ---> client
server ip :172.16.16.15,  port :5001 send to ---> client
server ip :172.16.16.15,  port :5000 send to ---> client
```

图 5.4(b) 对应场景服务器辅助测试结果

从收到的回复上来看，该内网主机也收到了两条来自于服务器 S 的回复，地址与上述结果相同。但因为两次 NAT 映射之后的公网地址的端口号不同，因此 NAT 类型为对称型 NAT，整个过程耗时 2s。

3、假设内网主机 A 作为首发端，并且选择了 UDP 通信模式，它将根据获取到的对端 NAT 类型和自身 NAT 类型选择合适的 NAT 穿越方案。

(1) 内网主机向公网节点进行注册

图 5.5(a) 内网主机获取自身与其他主机的注册信息

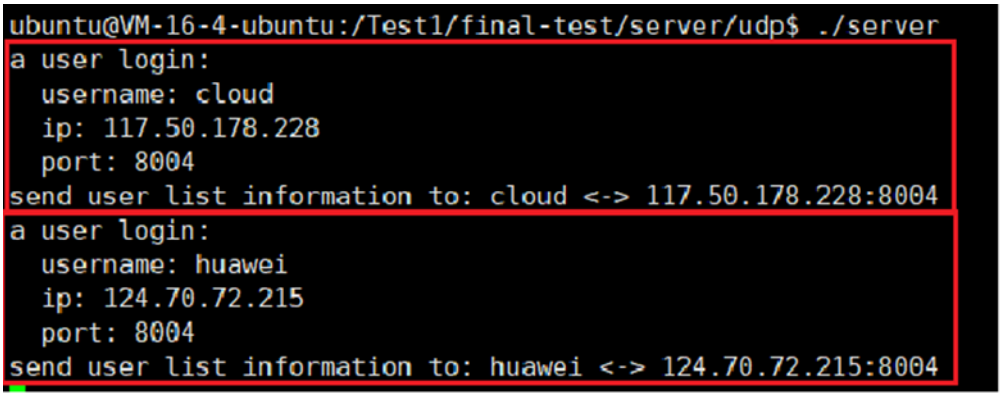

图 5.5(b) 公网主机辅助注册过程

从上述的结果从我们可以看出，当内网主机的客户端模块向公网节点的服务器模块发送登录消息之后，服务器模块会对其回复自身保存的所有注册用户信息，这些用户信息包括了每一个登录到服务器模块的主机节点的用户名、NAT 设备类型以及 NAT 映射后的公网地址信息。

(2) 两台端口限制锥型 NAT 下主机的 UDP 通信过程

图 5.7　两台端口限制锥型 NAT 下主机的 UDP 首次通信过程

上图 5.7 是上述登记在服务器上的两台端口对称型 NAT 下内网主机之间的首次通信过程。我们可以看到，主机“huawei”指定向对方主机“cloud”发送文本消息“66677788heu”，接下来“huawei”利用了服务器的辅助完成了对 NAT 的穿越，成功将消息送达“cloud”主机。整个 UDP 通信过程耗时 65ms，这其中绝大多数时间用在了 NAT 穿越之上。

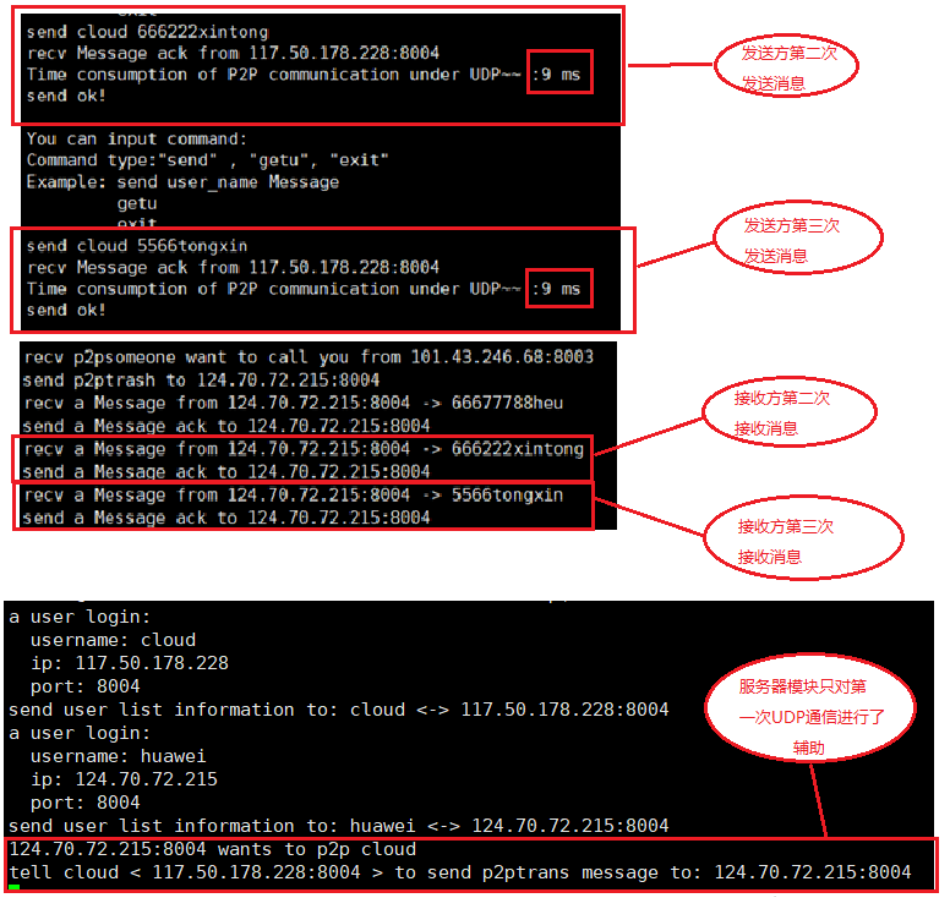

图 5.8　完成首次 NAT 穿越后的 UDP 通信过程

在完成了第一次通信，也即完成了 NAT 穿越之后，双方主机就可以通过心跳包维持“连接”，可以看到第二次和第三次通信过程只需 9ms 即可完成，原因是省去了 NAT 穿越环节，而是直接实现了点对点通信。而且从公网节点的服务器模块也可见它只对第一次 UDP 通信辅助进行 NAT 穿越，后续的 UDP 通信将不再进行参与。

(3) 一方为对称型 NAT 时的 UDP 通信过程

图 5.9　一方为对称型 NAT 时的 UDP 通信过程

从上图的分析可以看出，这是处于对称型 NAT 下的主机“ubuntu”向端口限制锥型 NAT 下的主机“cloud”发送 UDP 数据报“8899j”的过程，我们可以看到这里采用了改进后的端口预测方案，双方主机在 45ms 后完成了 NAT 穿越并完成了一次点对点通信。由于本实验中的对称型 NAT 为恒等对称型 NAT，测试环境是校园网 NAT 设备，端口的分配是针对于所有对外访问的全校师生，映射端口变化较快，心跳包每次发送都会使用一个新的端口号，导致心跳包功能在此场景下无法使用，

(4) 双对称型 NAT 是的 UDP 通信过程

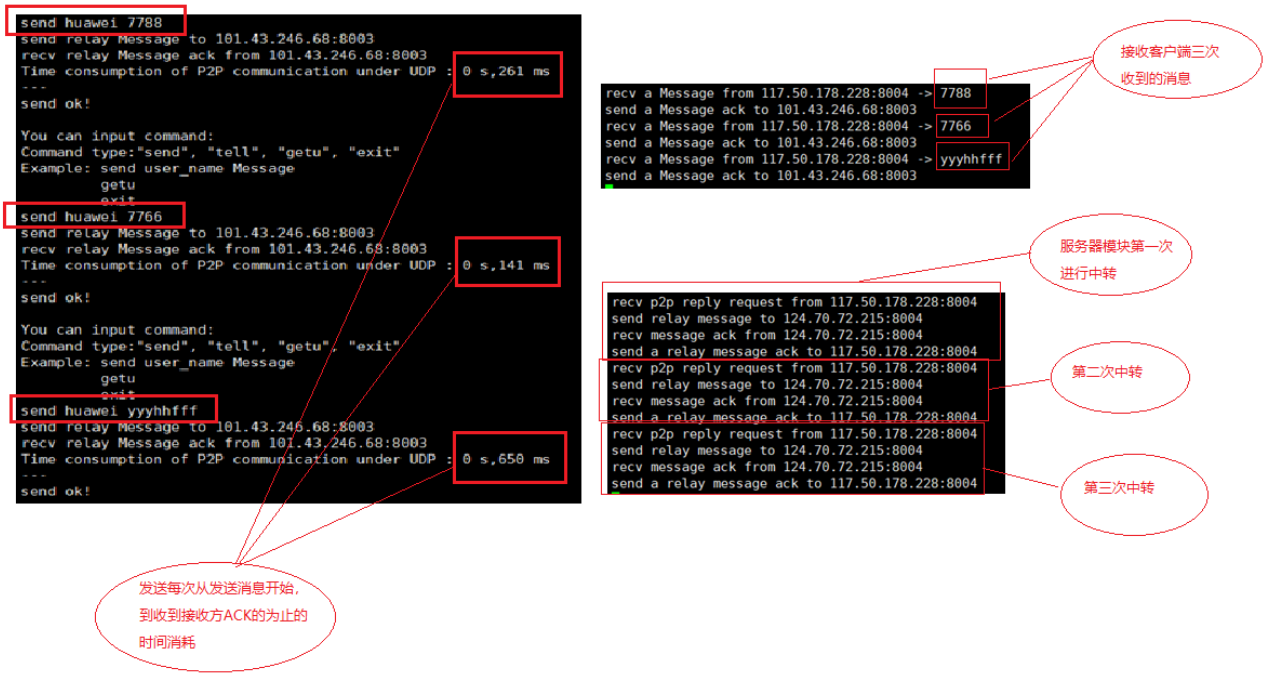

图 5.10　双对称型 NAT 时的 UDP 通信过程

上述图为双对称 NAT 场景下，在公网节点的服务器模块协助下完成 UDP 通信的过程。由于是双对称性 NAT 的情况，无法实现直接的 NAT 穿越而是由服务器模块对双方发送的消息进行中转。我们可以看到，发送方主机 A 向接收方主机 B 从发送消息开始到收到对方由服务器转发的 ACK 回应为止，每次需要消耗的时间分别为 261ms，141ms，650ms，消耗时间的长短取决于从主机 A 经公网节点再到主机 B 整条链路上的网络延迟情况，相对来说变化幅度还是比较大的。

(5) 实验结果分析

经统计，下表 5.1 给出了各种 UDP 环境下 NAT 穿越需要消耗的时间。

表 5.1　各种 UDP 环境下 NAT 穿越需要消耗的时间

	主机 A NAT 类型	主机 B NAT 类型	通信次数	平均消耗时间/s(20 次)
测试 1	端口限制锥型	端口限制锥型	1	57ms
测试 2	端口限制锥型	端口限制锥型	2,3,……	7ms
测试 3	端口限制锥型	对称型	1,2,3,……	56ms
测试 4	对称型	端口限制锥型	1,2,3,……	47ms
测试 5	对称型	对称型	1,2,3,……	461ms

注：这里限定主机 A 为消息的发送方，主机 B 为消息的接收方。对称型与对称型 NAT 之间的通信借由公网节点完全中继实现。其他锥型 NAT 由于实验条件的限制没有进行测试。

根据实验的统计结果可知：

① 针对于双方皆为锥型 NAT 的情况下，由于只需要在第一次 UDP 通信时进行 NAT 穿越，因此也只有第一次需要耗费比较长的时间，以后的第二、三、……次通信将是节点间的直接点对点通信，消耗的时间比第一次通信要少一个数量级。

② 针对于一方为对称型 NAT，一方为端口限制锥型 NAT。由于本实验中测试到的 NAT 网管为恒等对称型 NAT，双方主机在第一次通信完成之后无法通过心跳包维持“UDP 连接”，因此每次通信都必须重新进行 NAT 穿越。

③ 特殊的，针对于②中的场景，对称型 NAT 和端口限制锥型 NAT 担任的角色不同，在时间消耗上将略有不同：端口限制锥型 NAT 作为发送方时，需要向公网节点服务器模块发送一条 P2P_CHANGE 消息将自身变更为“被打洞方”，需要在服务器模块的协助下先作为被打洞方完成与接收方的 NAT 穿越，接着再向对方发送消息。整体上将会比对称型 NAT 直接作为发送方和打洞方要多消耗一段时间(<10ms)。

④ 最后是双方皆为对称型 NAT 的场景，此场景通过公网节点进行消息转发实现。可以看到其平均需要耗费的时间比需要完成一次 NAT 穿越的 UDP 通信多出一个数量级。

4、同样假设 A 为首发端，但选择了 TCP 通信模式，它也将根据对端和自身 NAT 类型采取合适的 NAT 穿越方案。

(1) 内网主机向公网节点进行注册

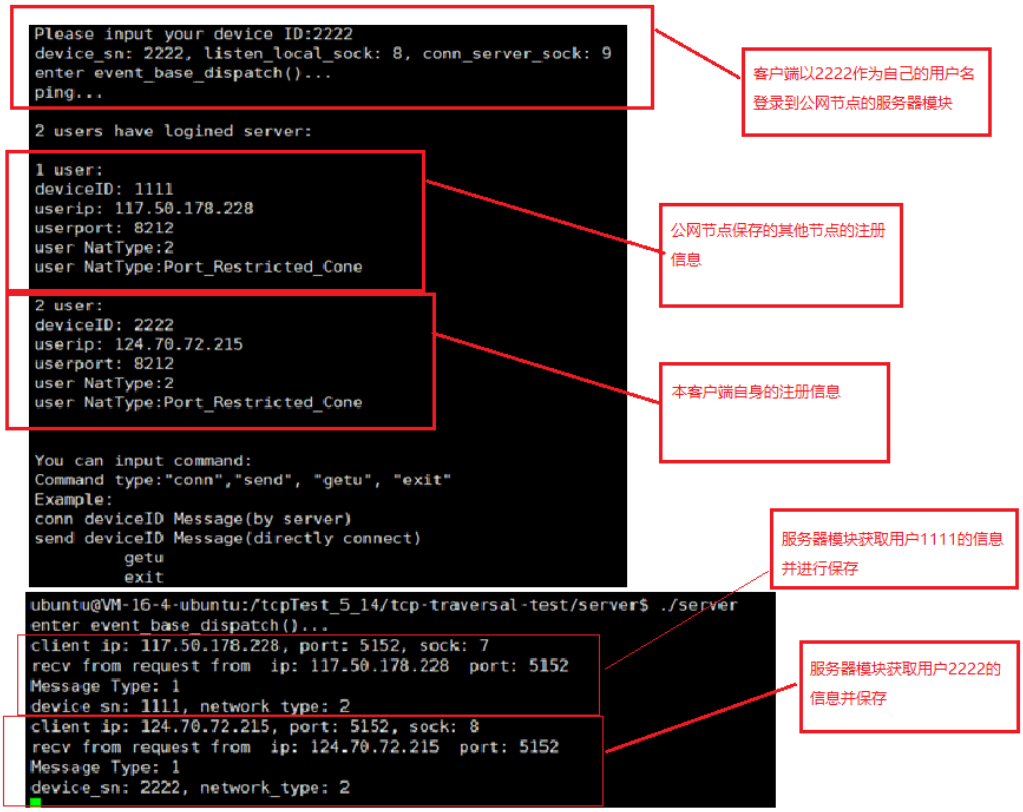

图 5.11　TCP 模式下内网主机登录与公网节点辅助过程示意图

从上图我们可以看到，内网主机“2222”作为第二个节点注册到了公网节点服务器模块上，公网节点获取其 NAT 映射地址与 NAT 类型的同时将其存储到本地的数据库中，并最终向其发送所有注册用户信息。

(2) 非对称性 NAT 下主机之间的 NAT 穿越

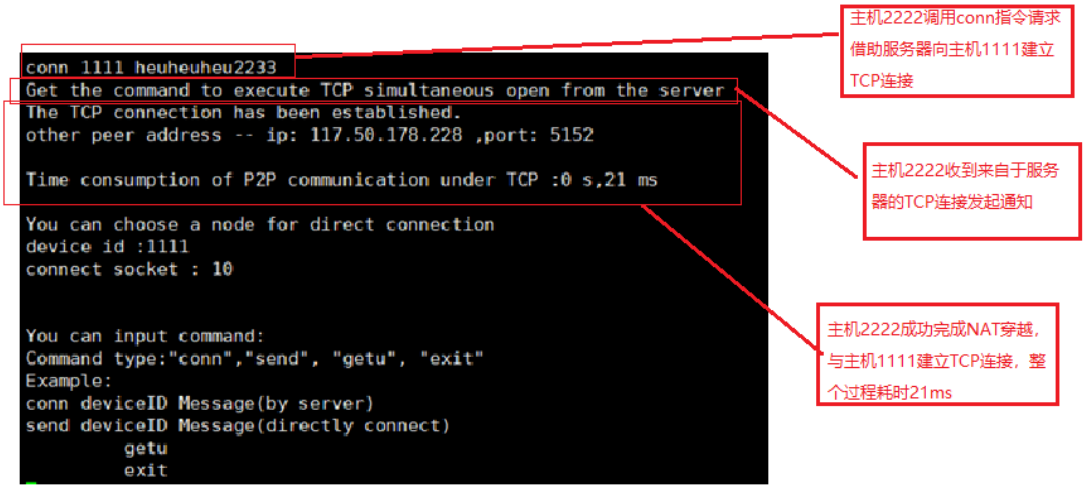

图 5.12(a) 非对称性 NAT 下主机之间的 NAT 穿越--连接发起方

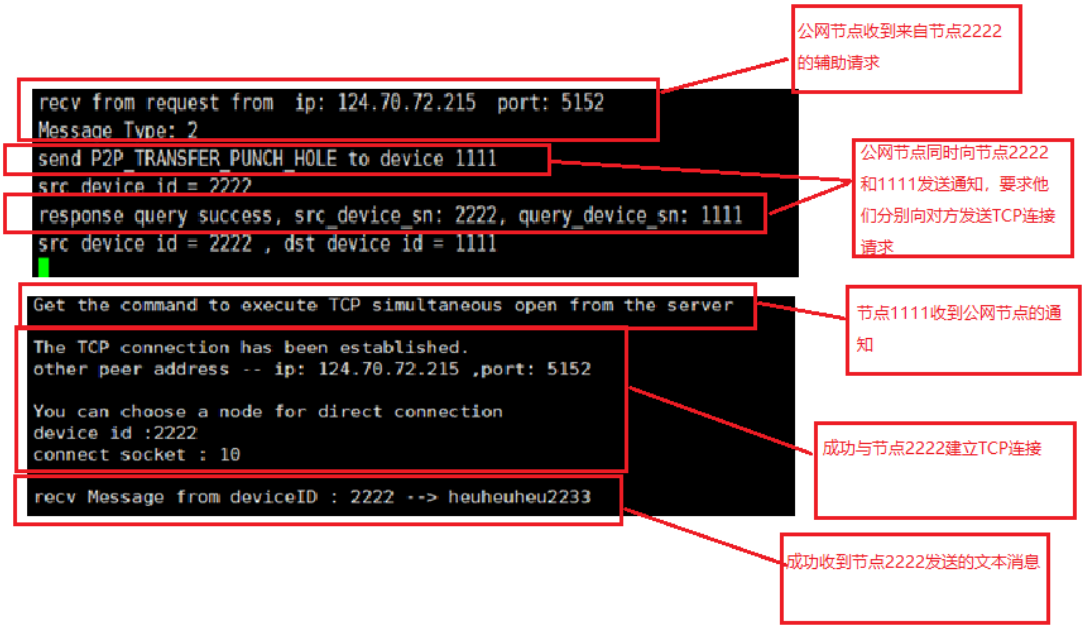

图 5.12(b) 非对称性 NAT 下主机之间的 NAT 穿越--公网节点与被连接方

上述两图展现了非对称型 NAT 之间一次完整的 NAT 穿越过程，这里主机“2222”试图借助公网节点完成 NAT 穿越，从而与主机“1111”建立 TCP 连接。公网节点在收到主机“2222”发送的辅助请求之后，将同时向“1111”和“2222”发送通知要求他们分别向对方发送 SYN 连接请求。整个 NAT 穿越过程耗费 21ms，完成了 TCP 连接的建立与文本消息“heuheuheu2233”的发送与接收。

(1) TCP 连接建立之后的数据收发

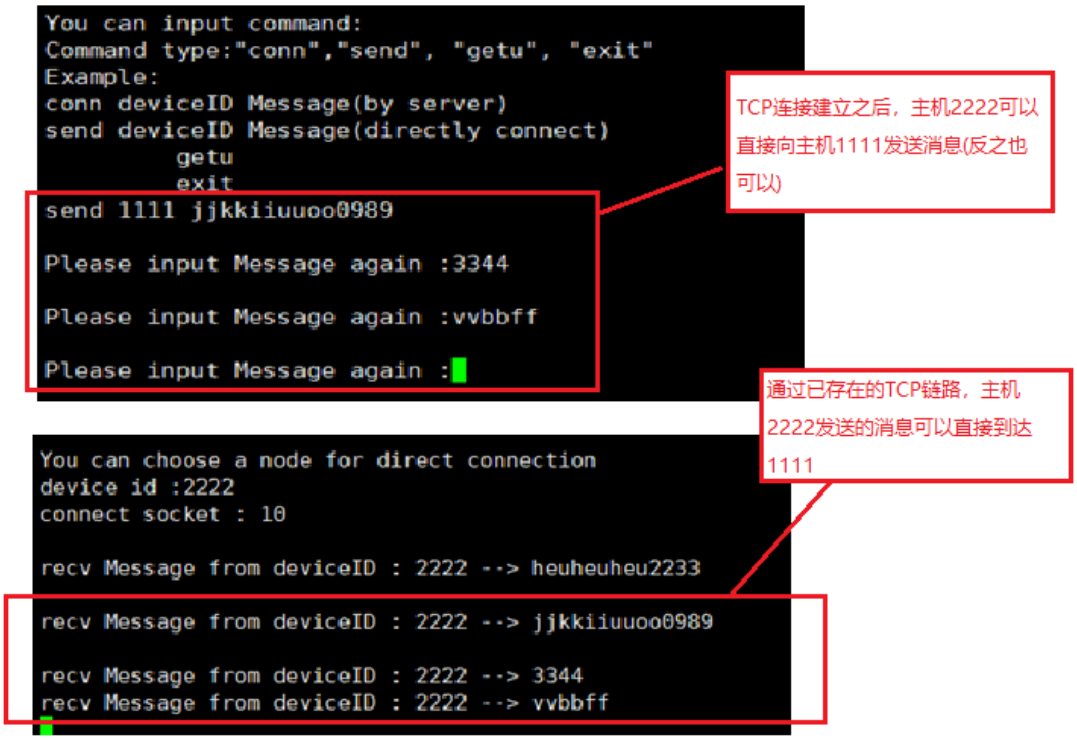

图 5.13　TCP 连接建立之后的消息收发

TCP 连接建立之后，无需任何公网节点的辅助，TCP 链路两端的节点可以向对方直接发送数据。只需调用 send 指令向对方发送消息，而且整个不过节点 1111 和节点 2222 都可作为数据的发送方也可以作为数据的接收方。

(2) 实验结果分析

经测试和统计，在 20 次实验过程中两台内网主机想要穿越 NAT 设备建立 TCP 连接，平均需要消耗 18ms 的时间。与 UDP 协议不同，由于 TCP 协议本身是面向连接的，因此对于 TCP 连接建立以后的数据收发是有保证的，因此对其所需时间不再进行统计

5.3 本章小结

本章主要介绍了本设计的测试环境，每一次测试包括两台内网主机与一台位于公网上的服务器。测试过程部分详细介绍了对于每个模块的使用方式和流程，以效果图的方式展现了整个设计中 NAT 穿越的可行性。最后针对 NAT 类型检测、UDP 下 NAT 穿越和 TCP 下 NAT 穿越各模块的时间性能进行了统计和分析，从结果上对理论进行了验证。

总　　结

IPv4 地址的短缺推动了 NAT 设备的普遍化，如今的互联网环境下几乎难以看到不使用 NAT 设备的网络主机。一方面，NAT 技术确实解决了地址不足的问题，但另一方面也严重阻碍了 P2P 技术的发展，这导致了许多依托于 P2P 网络的应用在不采取其他措施的情况下，只能部署于公网上的节点，亦或是部署于局域网中，区块链便是一个典型的例子。拥有从根本上解决这一问题的 IPv6 方案由于成本问题迟迟无法得到全面部署，作为过渡方案的 NAT 穿越便成为了人们研究的重点。当前存在的穿越技术多种多样，但任何一种技术都无法解决完美解决 P2P 通信的问题。

在本文中，首先介绍了 P2P 技术和 NAT 技术的基本原理；接着分析了各类 NAT 穿越方案简单的实现原理，结合实现难度与成本的问题最终选择了 STUN 方案作为研究的对象；在方案设计部分，先是介绍了现有 STUN 协议实现 NAT 穿越的原理和步骤，然后从理论上分析其性能，包括穿越过程所需的时间和流量，并以此为基础对 STUN 协议的 NAT 类型探测部分以及包含对称型设备时 NAT 穿越的端口预测环节进行了改进，减少其在 NAT 穿越时需要消耗的时间和流量，并在最终的测试环节验证了改进方案在实现 NAT 穿越上的可行性。

本设计针对于 UDP 通信和 TCP 通信分别给出了其 NAT 穿越的实现方案，同时本方案在设计时融合了 TURN 方案，针对于 STUN 无法实现穿越的场合进行了弥补，穿越的成功率相比于单纯的 STUN 方案有了相当的提升。本方案采用了模块化的设计方式，拥有良好的扩展性，具体将本设计应用到整个区块链系统中只需结合具体的需求进行些许改进，比如在区块链的 P2P 网络形成之初，通过节点选取算法找拥有公网 IP 的“超级节点”担任该设计中“服务器”的角色即可。

尽管本设计针对于大多数 NAT 穿越场景都能实现穿越，但针对于双对称型 NAT 场合或者是需要进行 TCP 通信但至少一方为对称型 NAT 的场合，无法解决 NAT 穿越问题，只能采用 TURN 方案借由公网节点完成中继，从结果上看尽管穿越的成功率得到了保证，但节点之间通信的时延是比较高的，更重要的是对于公网节点的依赖过大，相应的公网节点的负载也较重，因此如何设计一种更为灵巧的端口预测方案解决对称型 NAT 的穿越问题还需更深一步的研究；初次之外，本设计无法解决存在公有 NAT 设备的两主机之间的穿越问题，这一点也是未来研究中需要解决的问题之一。

参考文献

[1] Ethereum White Paper.A next-generation smart con-tract and decentralized application platform[EB/OL]. https://github.com/ethereum/wiki/wiki/White-Paper.2015.

[2] 黄秋波,安庆文,苏厚勤.一种改进 PBFT 算法作为以太坊共识机制的研究与实现[J].计算机应用与软件,2017,34(10):288-293+297.

[3] Drescher D. 区块链基础知识 25 讲[M]. 北京：人民邮电出版社, 2018-11-1.

[4] 孙盛源. 简述 NAT—网络地址转换[J]. 甘肃科技，2001, 5(7): 23-25.

[5] 高育滨. 基于 NAT 穿越的流媒体传输系统的设计与实现[D].华南理工大学,2019.DOI:10.27151/d.cnki.ghnlu.2019.003516.

[6] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann.2020. Mapping the Interplanetary Filesystem. https://arxiv.org/pdf/2002.07747.pdf.

[7] Liang Wang and Ivan Pustogarov. 2017. Towards Better Understanding of Bitcoin Unreachable Peers. (2017). http://arxiv.org/abs/1709.06837.

[8] 冯东煜,朱立谷,张雷,张迪,夏威夷,张瑞松.P2P 分布式存储中基于 UDT 的 NAT 穿越技术研究[J].计算机研究与发展,2014,51(S1):204-209.

[9] 陈永东. 基于多类型 NAT 的 TCP 穿透技术研究[D].四川师范大学,2016.

[10]胡军,周剑扬,师佳.P2P 网络中 UPnP 穿越 NAT 的研究与实现[J].现代计算机(专业版),2009(08):124-126.

[11]RFC 5766, Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)[S]. East Hanover: Rosenberg, 2010.

[12]张春红, 裘晓峰, 弭伟, 纪阳. P2P 技术全面解析[M]. 北京：人民邮电出版社, 2010.

[13]RFC 3489, STUN: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)[S]. J. Rosenberg, 2003.

[14]王止戈,彭宇峰,张苏灵,高传善.一种基于预测的 Symmetric NAT 穿越解决方案[J].计算机工程,2005(11):122-123+210.

[15]王勇,崔修涛,吕钊,李子成.基于探测对 Symmetric NAT 与端口受限 NAT 的穿透方案[J].计算机应用,2006(04):922-925.

[16]冯金哲,殷海兵.一种 Symmetric NAT 穿透的新方法[J].计算机应用与软件,2017,34(01):125-128.

[17]张伟欣,韩秀玲.TCP 穿透 NAT 的 P2P 通信技术研究[J].计算机时代,2008(11):62-64.

[18]何秀淼.TCP 穿越 NAT 的原理与实现方法[J].电信网技术,2014(07):86-89.

[19]孙君文. 基于 STUNT 协议的内网穿越解决方案的设计和实现[D].华东师范大学,2009.

[20]吕飞. 基于混合式 P2P 网络的 NAT 穿越技术的研究与应用[D].大连理工大学,2011.

[21]郑浩. 基于 STUN 协议的 NAT 穿越技术的研究与应用[D].武汉理工大学,2018.

[22]马义涛. 基于 P2P 网络应用的 NAT 穿越方案的分析与设计[D].上海交通大学,2008.

[23]甄世泉. TCP 穿越 NAT 的 P2P 通信关键技术研究与应用[D].北京邮电大学,2014.

[24]RFC 5389, Session Traversal Utilities for NAT (STUN)[S]. J. Rosenberg, 2008.

[25]邱耀群. 基于 STUN 协议的 NAT 穿越技术研究[D].宁波大学,2015.

[26]刘泽阳,徐武平.P2P 应用中一种多层 NAT 穿透解决方案的设计与实现[J].计算机应用,2011,31(07):1980-1983.

[27]胡国勇. 一种结合 P2P 及中转传输的网络用户文件分享系统[D].电子科技大学,2019.

[28]David Anderson. How NAT traversal works[EB/OL].https://tailscale.com/blog/how-nat-traversal-works/.2020

[29]蒙元胜. 基于 UDP/TCP 协议的 NAT 穿越方案研究[D].中山大学,2014.

[30]段志鸣. 基于混合式 P2P 网络 UDP 下 NAT 穿越方案的研究与设计[D].哈尔滨理工大学,2010.

攻读学士学位期间发表的论文和取得的科研成果

致　　谢

转眼间大学四年已经过去了，回顾这四年的大学生活不仅感慨良多。感谢我的母校哈尔滨工程大学为我提供如此优秀的学习平台。在这毕业之际，我由衷地向所有在生活与学业上帮助过我的老师和同学们表达感激。

首先，我衷心地感谢我的指导教师刘春鹏老师和曹傧老师，两位老师认真负责、治学严谨的态度给我留下了深刻的印象。每当在项目研究过程中遇到困难时，老师们总能提出宝贵的意见，使我拨云见日找到问题的所在。其次，我要感激曹明锐和冉越升两位师兄对我的无私帮助，在研究过程为我提供了许多建议，使我在研究过程中总能找到正确的方向。这里我还要感谢我的三位室友，边东泽、胡子聪和章子豪，他们认真好学的态度让寝室充满了积极向上的氛围，也推动并激励着为不断前行。

当然，我还要感谢我的父母，他们在生活上对我的支持与关怀成为我人生最强大的后盾，使得我能够顺利完成学业，真挚的感谢他们的无私之爱。

最后，感谢各位评审专家在百忙之中抽出时间对本文进行审阅，感谢你们提出的宝贵意见。