**华中科技大学网络空间安全学院**

2024秋网络安全课程设计报告

——Linux下状态检测防火墙的设计与实现

|  |  |
| --- | --- |
| 班 级： |  |
| 学 号： | U2021 |
| 姓 名： | MTX |
| 任课教师： | 王美珍 |

2024年11月15日

**报告评分表**

|  |  |  |  |
| --- | --- | --- | --- |
| **评分项目** | **分值** | **评分标准** | **得分** |
| 实验原理 | 15 | 15-13：统流程清晰，报文处理过程描述清楚；  12-10：系统流程比较清晰，报文处理过程描述比较清楚；  9-0：描述简单 |  |
| 系统设计 | 15 | 15-13：系统结构设计描述详细、清楚、完整，前后关系清晰；  12-10：系统结构设计比较清楚，关键模块、流程等都进行了描述；  9-0：系统结构设计描述比较简单或不完整 |  |
| 详细设计 | 15 | 15-13：关键模块设计描述详细、清楚、完整，前后关系清晰；  12-10：关键模块设计比较清楚，关键模块、流程等都进行了描述；  9-0：关键模块设计描述比较简单或不完整 |  |
| 问题分析与解决 | 20 | 20-16：问题描述准确，原因分析清晰，解决方法正确；  15-12：问题描述较准确，原因分析较清晰，解决方法较合理；  11-0：问题描述简单，原因分析不清，解决方法不合理； |  |
| 系统测试 | 15 | 15-13：任务完成，针对任务点的测试，对结果有分析  12-10：针对任务点的测试截图，没分析  9-0：测试很简单，没有覆盖任务点 |  |
| 心得体会 | 10 | 10-8：体会真实具体  7-5：体会比较空洞  4-0：没有写什么体会 |  |
| 格式规范 | 10 | 图、表的说明，行间距、缩进、目录等，一种不规范扣1分 |  |
| **总 分** | | |  |
| **评分人：** | | | |

**目 录**

[一. 实验目的和要求 1](#_Toc182496202)

[1.1. 目的 1](#_Toc182496203)

[1.2. 要求 1](#_Toc182496204)

[二. 实验原理（简述） 1](#_Toc182496205)

[三. 实验环境和采用的工具 1](#_Toc182496206)

[四. 系统设计 2](#_Toc182496207)

[4.1. 系统结构设计 2](#_Toc182496208)

[4.1.1. 内核模块设计 (my\_firewall.c) 2](#_Toc182496209)

[4.1.2. 用户交互程序设计 (UI.c) 2](#_Toc182496210)

[4.1.3. 日志程序设计 (log.c) 3](#_Toc182496211)

[4.2. 核心数据流程 3](#_Toc182496212)

[4.2.1. 网络报文数据 3](#_Toc182496213)

[4.2.2. 防火墙规则数据 3](#_Toc182496214)

[4.2.3. NAT规则数据 3](#_Toc182496215)

[4.2.4. TCP报文数据流动 3](#_Toc182496216)

[4.2.5. UDP报文数据流动 4](#_Toc182496217)

[4.2.6. ICMP报文数据流动 4](#_Toc182496218)

[五. 详细设计实现 5](#_Toc182496219)

[5.1. 关键模块流程 5](#_Toc182496220)

[5.1.1. 内核规则匹配模块 5](#_Toc182496221)

[5.1.2. 内核状态检测模块 7](#_Toc182496222)

[5.1.3. 内核NAT模块（IN） 9](#_Toc182496223)

[5.1.4. 内核NAT模块（OUT） 11](#_Toc182496224)

[5.2. 关键数据结构 14](#_Toc182496225)

[5.2.1. 连接表结构 (Connect\_list) 14](#_Toc182496226)

[5.2.2. 规则表结构 (Rule\_list) 14](#_Toc182496227)

[5.2.3. NAT规则表结构 (NAT\_list 和 NAT\_connect\_list) 15](#_Toc182496228)

[5.3. 模块接口设计 16](#_Toc182496229)

[5.3.1. 防火墙过滤接口 16](#_Toc182496230)

[5.3.2. NAT处理接口 16](#_Toc182496231)

[5.3.3. Netlink 通信接口 17](#_Toc182496232)

[5.3.4. 日志记录接口 17](#_Toc182496233)

[5.3.5. 校验和更新接口 17](#_Toc182496234)

[六. 问题分析与解决 18](#_Toc182496235)

[6.1. 内核卸载后卡死问题 18](#_Toc182496236)

[6.1.1. 问题描述 18](#_Toc182496237)

[6.1.2. 原因分析 18](#_Toc182496238)

[6.1.3. 解决方法 18](#_Toc182496239)

[6.2. 内核卸载后卡死问题 18](#_Toc182496240)

[6.2.1. 问题描述 18](#_Toc182496241)

[6.2.2. 原因分析 18](#_Toc182496242)

[6.2.3. 解决方法 18](#_Toc182496243)

[6.3. 内核卸载后卡死问题 19](#_Toc182496244)

[6.3.1. 问题描述 19](#_Toc182496245)

[6.3.2. 原因分析 19](#_Toc182496246)

[6.3.3. 解决方法 19](#_Toc182496247)

[6.4. 使用NAT后UDP接收报文就退出问题 19](#_Toc182496248)

[6.4.1. 问题描述 19](#_Toc182496249)

[6.4.2. 原因分析 19](#_Toc182496250)

[6.4.3. 解决方法 19](#_Toc182496251)

[七. 系统测试 20](#_Toc182496252)

[7.1. 测试环境 20](#_Toc182496253)

[7.2. 功能测试 20](#_Toc182496254)

[7.2.1. UDP拦截测试 20](#_Toc182496255)

[7.2.2. TCP拦截测试 23](#_Toc182496256)

[7.2.3. ICMP拦截测试 25](#_Toc182496257)

[7.2.4. 状态检测测试 25](#_Toc182496258)

[7.2.5. NAT功能测试 25](#_Toc182496259)

[7.3. 性能测试 28](#_Toc182496260)

[八. 心得体会及意见建议 28](#_Toc182496261)

[8.1. 心得体会 28](#_Toc182496262)

[8.2. 意见建议 29](#_Toc182496263)

**课程设计报告要求：**

**1. 报告不可以抄袭，发现雷同者记为0分；**

**2. 报告中不可以只粘贴大段代码，应是文字与图、表结合的，需要说明流程的时候，也应该用流程图或者伪代码来说明；如果发现有大段代码粘贴者，报告打回重写；**

**3. 报告格式要求规范。**

# 实验目的和要求

## 目的

结合理论课程学习，深入理解计算机网络安全的基本原理与协议，巩固计算机网络安全基本理论知识。

熟练掌握计算机网络编程方法，拓展学生的应用能力。

加强对网络协议栈的理解。

提高分析、设计软件系统以及编写文档的能力。

锻炼表达和交流沟通能力。

## 要求

正确理解题意。

具有良好的编程规范和适当的注释。

有详细的文档，文档中应包括课题涉及的基础知识、设计思路、程序流程图、程序清单、开发中遇到的问题及解决方法、设计中待解决的问题及改进方向。

# 实验原理（简述）

基于Linux的Netfilter框架，设计一个具备状态检测功能的防火墙，以有效过滤和管理网络通信的数据包。防火墙通过维护状态检测表（连接表），记录通过的连接状态，包括源地址、目的地址、协议、端口信息及连接状态。对于每个进入系统的包，首先查找连接表，若该包属于现有连接，则直接处理，否则通过规则匹配决定是否允许连接。此外，该系统支持NAT转换和协议分析，能够实现源或目的地址的转换，保证网络数据的正常流通。

# 实验环境和采用的工具

本实验在Linux操作系统环境下进行，主要采用Ubuntu20.04作为实验系统。使用的核心工具包括Linux内核自带的Netfilter框架，用于实现数据包过滤和状态检测功能。开发工具采用GCC编译器进行内核模块和应用程序的编译，使用Makefile管理编译流程，以便于模块的插入和卸载。调试工具主要利用dmesg命令查看内核日志输出，以便调试内核模块运行情况。此外，为实现用户空间和内核空间的通信，实验中使用字符设备或Netlink通信接口，方便用户空间程序与防火墙内核模块的交互操作。

# 系统设计

本实验系统设计由内核模块my\_firewall.c和用户交互程序UI.c以及一个负责保存日志的log.c组成，通过Netlink实现内核与用户空间的通信。

## 系统结构设计

系统结构设计见图4-1。各模块的详细介绍见4.1.1 - 4.1.3。

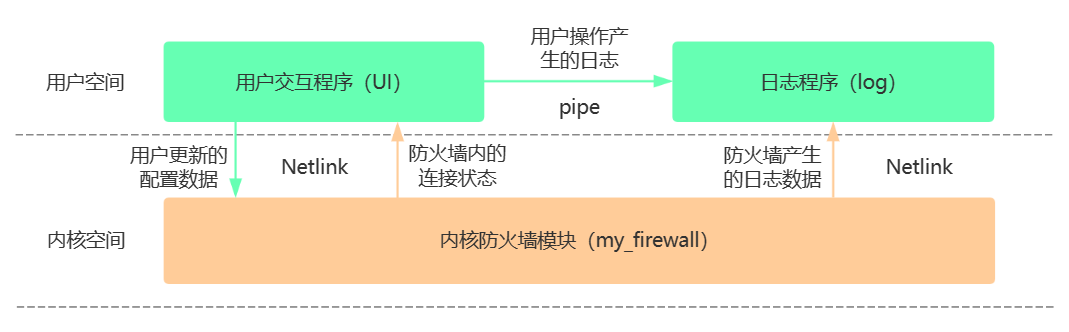


图4-1 系统结构图

### 内核模块设计 (my\_firewall.c)

内核模块的主要功能是通过Netfilter拦截并检查网络报文，决定是否放行。模块具体实现以下功能：

**规则过滤：**模块利用Netfilter的hook函数捕获网络包，调用rule\_match()函数逐条检查是否符合规则。符合规则的包将放行（NF\_ACCEPT），否则丢弃（NF\_DROP）。

**状态检测：**针对TCP、UDP和ICMP协议的报文，内核模块维护连接状态表，用于追踪通信状态，并在符合条件时更新或删除连接状态。

**NAT转换：**模块支持NAT功能，应用于内部和外部地址的转换，并通过Netlink接口接收用户程序的配置更新NAT表，同时通过Netlink将已经建立的NAT连接发送给用户空间进行查看。

**日志生成：**模块通过检测用户的设置，判断是否需要生成日志，若需要，则按照特定格式生成日志后，利用Nrtlink发送给用户空间的log程序。

### 用户交互程序设计 (UI.c)

用户交互程序用于提供命令行接口，接收用户输入的防火墙规则和NAT配置，并通过Netlink接口传递到内核模块。它实现了以下主要功能：

**规则管理：**用户通过命令行输入规则（包括源/目的IP、端口、协议、动作等），程序将规则写入配置文件，并通过Netlink接口将规则传递至内核。

**NAT配置管理：**用户程序读取和解析NAT配置文件，将配置转换为适合内核处理的格式后，通过Netlink接口更新内核的NAT表。

**模块控制：**用户可以通过命令加载和卸载内核模块，实现对防火墙的启动与停止​。

### 日志程序设计 (log.c)

用户空间的日志程序会接收内核空间发来的日志信息，并保存在文件中。

## 核心数据流程

### 网络报文数据

在my\_firewall.c中，telnet\_filter函数通过Netfilter钩子函数捕获每一个经过的网络报文（skb数据结构），并提取其IP、协议和端口信息，用于后续的规则匹配和状态检查。

### 防火墙规则数据

UI.c读取规则文件（rule），用户可以通过命令行交互来添加或修改规则。每条规则包含源IP、目的IP、协议和动作等信息。规则信息通过Netlink接口传递到内核模块my\_firewall.c，并由rule\_match函数用于逐条匹配。如果规则符合条件，将放行该报文并记录日志；否则，报文会被丢弃。规则匹配的处理逻辑采用五元组匹配，使系统具备高效的过滤能力​。

### NAT规则数据

NAT规则由UI.c通过Netlink接口传入my\_firewall.c，并存储于NAT表结构中。当捕获到符合NAT规则的数据包时，my\_firewall.c会依据NAT表中的源IP或端口进行转换，之后更新数据包的源地址和目的地址，重新计算校验和并放行。例如，telnet\_NAT\_in和telnet\_NAT\_out函数对入站和出站NAT包进行处理，实现内部和外部地址转换功能​。

**NAT规则数据**和**防火墙规则数据**的流动，见4.1.系统结构设计中的系统结构图4-1。

### TCP报文数据流动

在my\_firewall.c中，TCP连接状态机在接收到SYN包时为新连接，通过匹配防火墙规则（rule\_list），如果符合放行规则，则创建状态条目，并将连接的状态信息存储在状态表（Connect\_list结构）中。后续的TCP包将匹配该状态条目，直接放行或更新状态，如SYN-ACK或FIN等待。telnet\_filter函数调用connect\_match函数检查连接状态是否有效，以便丢弃不符合状态的TCP包。同时连接存在时间限制，如果该连接一段时间没有数据流，则会从连接表中删除。具体数据流动见图4-2。

### UDP报文数据流动

UDP属于无连接协议，系统为其实现虚拟连接追踪。每个UDP连接在状态表中会创建一个临时条目，允许返回的数据包通过；超时则删除条目。UI.c的规则管理功能支持将UDP配置到规则表中，并在my\_firewall.c中通过connect\_match来管理临时的UDP连接状态​。同时连接存在时间限制，如果该连接一段时间没有数据流，则会从连接表中删除。具体数据流动见图4-2。

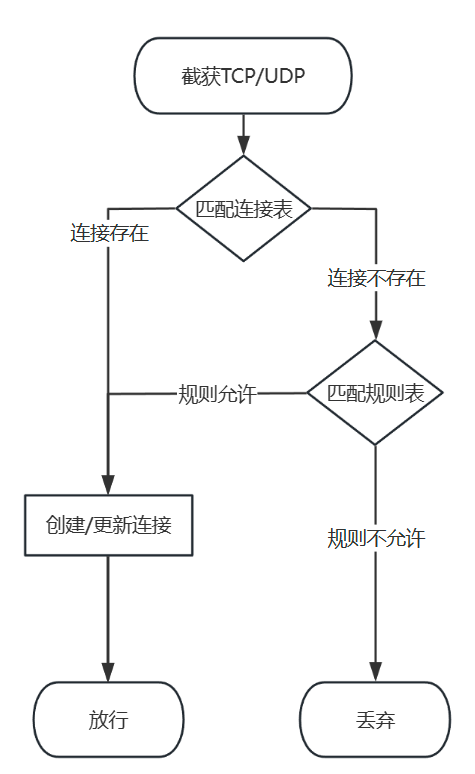


图4-2 TCP/UDP数据流动

### ICMP报文数据流动

ICMP状态机主要处理如ping（echo）请求及其回复。系统通过Netfilter hook捕获ICMP报文，并在状态表中记录其ID、源IP和目的IP。在接收到回复包时，telnet\_filter会查找对应的请求条目，确认每一个echo都有对应的request请求后才后放行；如果未找到对应条目或超时，则丢弃包。此机制确保仅合法ICMP通信能通过防火墙​。具体数据流动见图4-3。

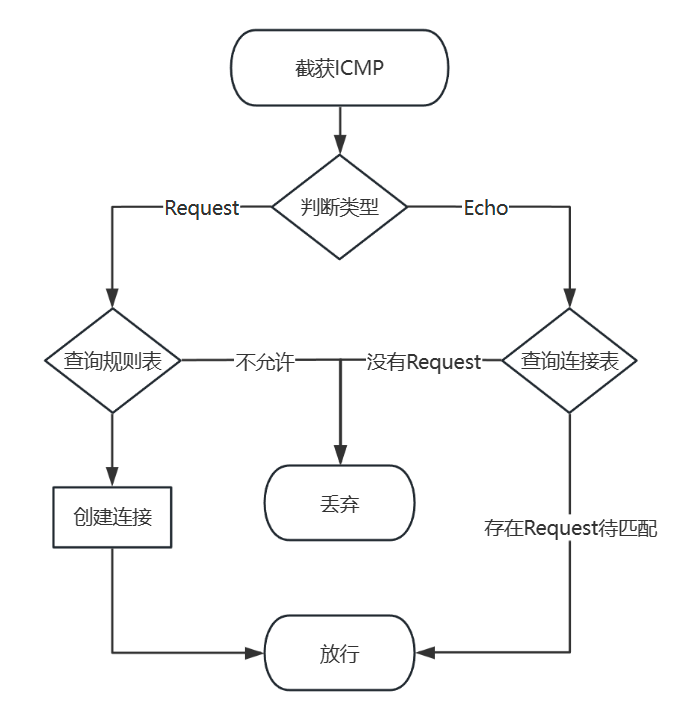


图4-3 ICMP数据流动

# 详细设计实现

## 关键模块流程

### 内核规则匹配模块

内核规则匹配模块的主要功能是检查一个数据包是否符合已配置的防火墙规则。函数通过匹配数据包的五元组信息（源 IP、目的 IP、源端口、目的端口、协议类型）来确定是否放行数据包，并根据匹配的规则创建新的连接条目，便于后续的数据包自动放行。

操作流程

1. 提取数据包的 IP 头部信息：通过 ip\_hdr(skb) 获取 IP 头部信息，提取出数据包的协议类型（iph->protocol）、源 IP（iph->saddr）、目的 IP（iph->daddr）。
2. 逐条遍历规则列表 rule\_list：函数初始化 i=1 并进入循环，逐条遍历 rule\_list 中的规则条目，检查当前数据包是否符合规则。
3. 检查协议、源 IP 和目的 IP 匹配：
4. 对于每条规则，函数检查数据包的协议类型、源 IP 和目的 IP 是否与规则条目相匹配。支持的匹配条件包括：

完全匹配。

通配（规则中使用 0xffffffff 表示任意协议、任意源 IP 或任意目的 IP）。

1. 如果协议和 IP 匹配，则继续检查端口信息。
2. TCP/UDP 协议的端口匹配：
3. 如果协议为 TCP 或 UDP，函数从 IP 头后获取 TCP/UDP 头部，提取源端口（sport）和目的端口（dport）。
4. 检查数据包的源端口和目的端口是否与规则条目中的端口相匹配，支持精确匹配和通配（0xffffffff 表示任意端口）。
5. 若端口匹配成功，则继续检查规则动作。
6. 匹配成功后的操作：
7. 允许规则 (rule\_list[i].rule 为真)：

函数为该连接创建一个新的连接条目 new\_connect1，记录源 IP、目的 IP、源端口、目的端口、协议类型以及超时时间等信息。

将该条目插入连接表 Connect\_index，确保后续的数据包可以直接放行。

如果规则要求记录日志（rule\_list[i].log 为真），调用 print\_log() 输出日志。

返回 1，表示放行该数据包。

1. 拒绝规则 (rule\_list[i].rule 为假)：

函数直接返回 0，表示丢弃数据包。

如果要求记录日志，调用 print\_log() 输出拒绝日志。

1. ICMP 协议的处理：
2. 对于 ICMP 协议，检查源 IP 和目的 IP 是否匹配规则。
3. 若匹配成功且规则允许，创建一个新的 ICMP 连接条目并插入 Connect\_index，用于匹配后续的 ICMP 包。
4. 记录日志（如 rule\_list[i].log 为真），然后返回 1 表示放行。
5. 未匹配的规则：如果所有规则均未匹配，返回 0，表示该数据包不符合任何允许的规则，不予放行。具体实现的流程见图5-1。

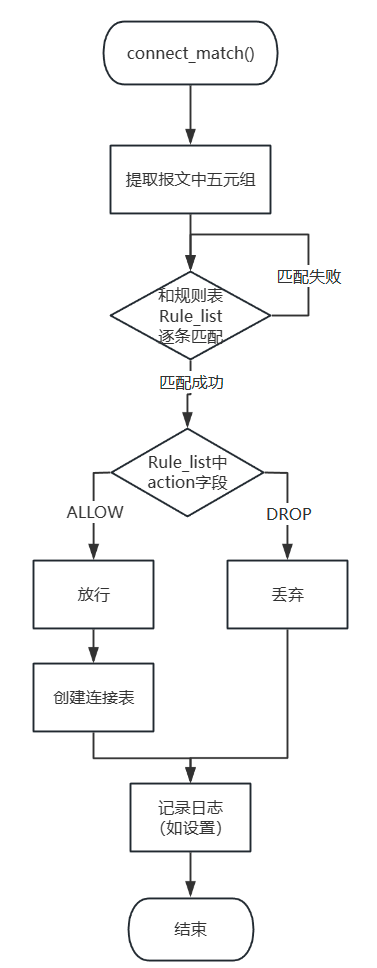


图5-1内核规则匹配模块流程图

### 内核状态检测模块

内核状态检测模块主要用于判断一个传入的数据包是否符合已存在的连接条目，并根据匹配结果决定是否放行数据包。它依据五元组信息（源IP、目的IP、源端口、目的端口、协议类型）来检测数据包的合法性，并在匹配后更新连接超时。

操作流程

1. 提取IP头部信息：函数首先通过 ip\_hdr(skb) 从 skb 数据包中提取 IP 头部信息，得到协议类型（iph->protocol）、源 IP（iph->saddr）和目的 IP（iph->daddr）。
2. 计算哈希索引：将源IP和目的IP右移16位后取低16位，进行异或运算得到 index1^index2。该索引用于快速定位 Connect\_list 中可能的连接条目链表。
3. 遍历连接条目链表：通过 Connect\_index[index1^index2] 获取哈希索引对应的连接条目链表，并开始逐条遍历：
4. 超时检查：对于每个连接条目，检查其超时时间（p->time）。若条目已超时（即当前时间大于条目时间），调用 delete\_connect() 函数将其删除。
5. 协议类型匹配：确认条目协议类型与数据包协议类型是否一致（p->type == type），确保协议一致性。
6. TCP/UDP连接匹配：
7. 提取端口信息：对于 TCP 和 UDP 数据包，从 IP 头后提取 TCP/UDP 头部，获取源端口（sport）和目的端口（dport）。
8. 匹配五元组信息：检查五元组信息是否与连接条目一致。匹配方式为：

源IP和源端口、目的IP和目的端口与连接条目正向匹配。

或者源IP和源端口、目的IP和目的端口与连接条目反向匹配。

1. 更新超时和日志记录：如果匹配成功，将该连接的超时时间更新为当前时间加上预定义的超时时间 CONNECT\_STATUE\_TIME。若 log 标志被设置，则调用 print\_log() 记录日志信息。最后返回 1 表示放行该数据包。
2. ICMP连接匹配：如果协议为 ICMP，检查包的源IP、目的IP是否与条目一致。若匹配成功：
3. 删除连接条目，确保 ICMP 请求不会重复出现。
4. 若 log 标志被设置，则记录日志。
5. 返回 1 表示放行。
6. 无匹配结果：若所有条目均未匹配成功，函数返回 0，表示该数据包不属于已允许的连接，不予放行。

具体流程介绍见图5-2。

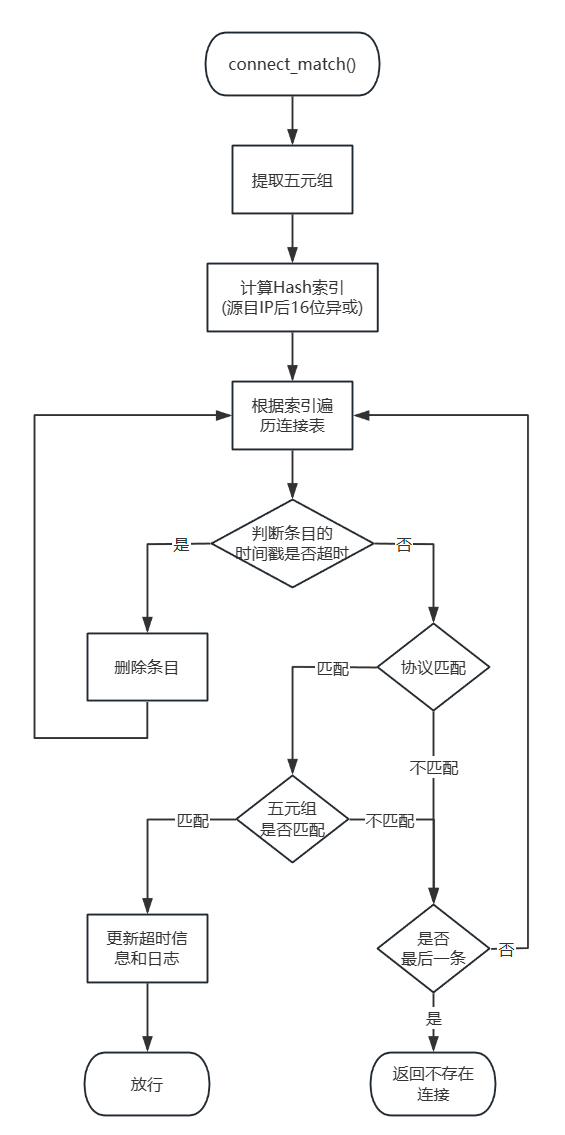


图5-2内核状态检测模块流程图

### 内核NAT模块（IN）

内核NAT模块（IN）用于处理入站 NAT 转换，将外部 IP 地址和端口转换回本地网络的私有 IP 地址和端口，以确保数据包能够被正确路由到内部的目标主机。

操作流程：

1. 捕获数据包
2. 通过 Netfilter 的钩子机制捕获所有入站的数据包。
3. 函数通过 ip\_hdr(skb) 提取 IP 头信息，获取源 IP (saddr)、目的 IP (daddr) 和协议类型 (type)。
4. 查找 NAT 连接条目
5. 对于每个入站的数据包，首先计算索引 (((saddr >> 16) & 0xffff) ^ sport) 并在 NAT\_connect\_index 中查找 NAT 条目。
6. 如果找到匹配的 NAT 条目，则执行 NAT 转换：

将数据包的目的 IP 和目的端口替换为 NAT 表中对应的内部 IP 和端口。

更新 NAT 条目的超时时间，表示该连接仍处于活动状态。

调用 update\_checksum(skb) 更新数据包的校验和。

1. 处理不同协议的数据包
2. TCP 和 UDP：针对 TCP 和 UDP 协议，从 IP 头部后提取相应的传输层头部，获取源端口 (sport) 和目的端口 (dport)，并根据五元组信息匹配 NAT 条目。
3. ICMP：对于 ICMP 数据包，匹配 ICMP id 字段，确保能够正确还原 ICMP 请求和响应包的 IP 地址。
4. 记录日志并放行数据包
5. 成功进行 NAT 转换后，通过 printk 打印日志信息，记录转换后的 IP 地址和端口。
6. 返回 NF\_ACCEPT 以允许数据包通过。

内核NAT模块（IN）具体流程见图5-3。

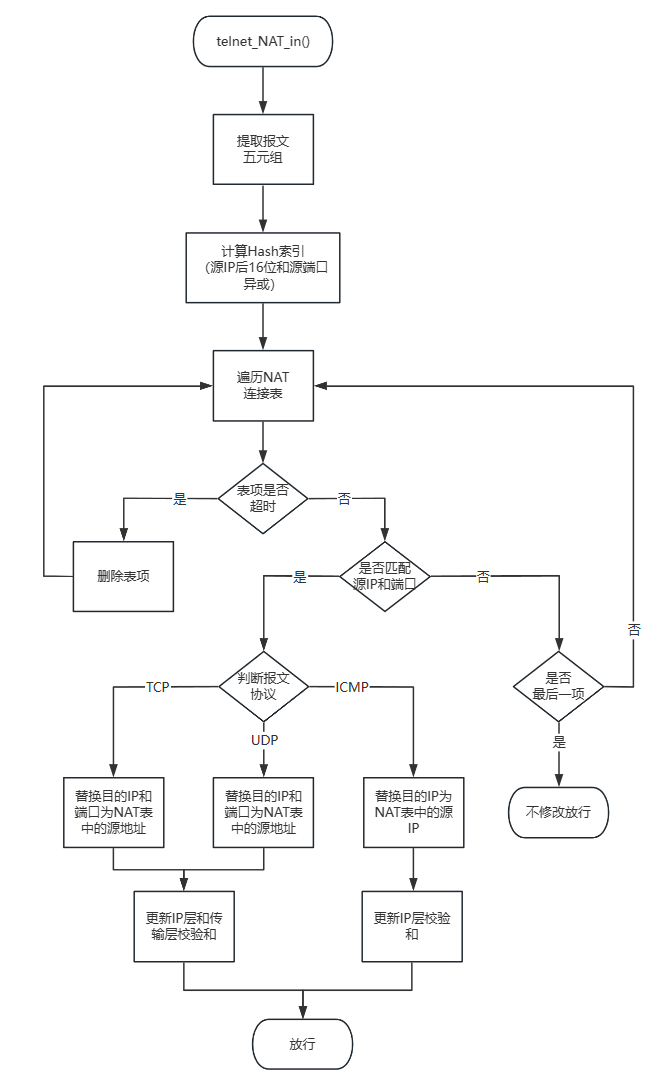


图5-3内核NAT模块（IN）模块流程图

### 内核NAT模块（OUT）

内核NAT模块（OUT）负责处理出站 NAT 转换，将私有 IP 地址和端口转换为公共 IP 地址和外部端口，使得来自本地网络的数据包能够通过 NAT 转换出站。

操作流程：

1. 捕获数据包
2. 该函数通过 Netfilter 的钩子机制捕获所有出站的数据包，包括 TCP、UDP 和 ICMP 协议的数据包。
3. 函数首先通过 ip\_hdr(skb) 提取 IP 头信息，获取源 IP (saddr)、目的 IP (daddr) 和协议类型 (type)。
4. 查找现有 NAT 连接条目
5. 根据数据包的目的 IP 和目的端口生成哈希索引 (((daddr >> 16) & 0xffff) ^ dport)，用于在 NAT\_connect\_index 中查找是否存在匹配的 NAT 连接条目。
6. 如果找到匹配的 NAT 条目，说明这是一个已有的 NAT 连接：

使用 NAT 表中的外部 IP 地址和外部端口替换数据包的源 IP 地址和端口。

更新 NAT 条目的超时时间，以表示该连接仍处于活动状态。

最后调用 update\_checksum(skb) 更新数据包的校验和，并返回 NF\_ACCEPT 允许数据包通过。

1. 新建 NAT 连接条目：如果找不到匹配的 NAT 条目，说明这是一个新的连接：
2. 遍历 nat\_list 查找与源 IP 匹配的 NAT 规则。
3. 根据匹配到的规则，生成一个外部随机端口（通过 generate\_random\_port() 函数）。
4. 创建一个新的 NAT 连接条目，将源 IP (saddr) 和源端口 (sport) 关联到外部 IP 和外部端口，并将该条目插入 NAT\_connect\_index 中。
5. 更新数据包的源 IP 和源端口，确保经过 NAT 转换后能够正确通过外部网络。
6. 调用 update\_checksum(skb) 更新校验和。
7. 记录日志并放行数据包
8. 在匹配成功或创建新 NAT 条目后，函数通过 printk 打印 NAT 日志信息，包括转换前后的 IP 地址和端口。
9. 最后返回 NF\_ACCEPT 以允许数据包通过。

内核NAT模块（OUT）具体流程见图5-4。

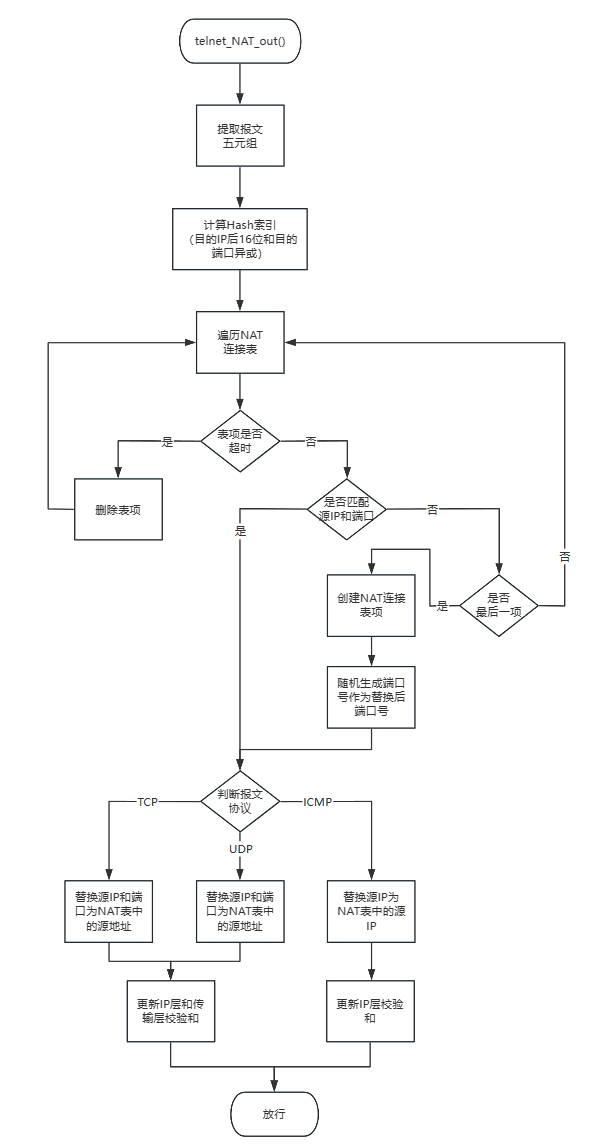


图5-4内核NAT模块（OUT）模块流程图

## 关键数据结构

### 连接表结构 (Connect\_list)

|  |  |
| --- | --- |
| **表5-1 Connect\_list结构体** | |
| 1 | typedef struct connect\_list { |
| 2 | unsigned int src\_ip; |
| 3 | unsigned int src\_port; |
| 4 | unsigned int des\_ip; |
| 5 | unsigned int des\_port; |
| 6 | unsigned int type; |
| 7 | unsigned int log; |
| 8 | unsigned long long time; |
| 9 | struct connect\_list \*next; |
| 10 | struct connect\_list \*last; |
| 11 | } Connect\_list; |

Connect\_list结构存储系统中活跃连接的状态，记录每个连接的源地址、目标地址、端口及协议类型，以便快速匹配后续报文。该结构的设计包括以下字段：

src\_ip和des\_ip：源和目的IP地址，唯一标识连接的两个端点。

src\_port和des\_port：源和目的端口号（TCP/UDP协议），用于端口级别的连接识别。

type：协议类型（如TCP或UDP），以支持不同协议的状态管理。

time：记录连接的超时时间，以便系统自动清理超时连接。

next和last：用于形成双向链表结构，使状态表中的条目可以进行快速插入和删除操作​，通过源地址和目的地址的组合来减少查找时间，加快连接匹配速度。

### 规则表结构 (Rule\_list)

|  |  |
| --- | --- |
| **表5-2 Rule\_list结构体** | |
| 1 | typedef struct rule\_list { |
| 2 | unsigned int src\_ip; |
| 3 | unsigned int src\_port; |
| 4 | unsigned int des\_ip; |
| 5 | unsigned int des\_port; |
| 6 | unsigned int type; |
| 7 | unsigned int rule; |
| 8 | unsigned int log; |
| 9 | }Rule\_list; |

Rule\_list结构定义了每条规则的必要信息，包括源/目的IP、端口、协议类型和行为。这些字段帮助防火墙快速匹配传入或传出的报文，决定其是否被放行或丢弃。结构包含以下主要字段：

src\_ip 和 des\_ip：源IP和目的IP地址，用于识别数据包的源和目标网络位置。

src\_port 和 des\_port：源端口和目的端口，适用于TCP/UDP协议，帮助识别具体的应用。

type：协议类型（如TCP、UDP、ICMP），用于区分不同的网络层协议。

rule：指定对匹配的数据包执行的操作（如ALLOW或DROP）。

log：标志是否记录该规则触发的报文日志，用于后续审计和排查​

### NAT规则表结构 (NAT\_list 和 NAT\_connect\_list)

|  |  |
| --- | --- |
| **表5-3 Rule\_list结构体** | |
| 1 | typedef struct nat\_list { |
| 2 | unsigned int inner\_ip; |
| 3 | unsigned int outside\_ip; |
| 4 | }NAT\_list; |
| 5 |  |
| 6 | typedef struct nat\_connect\_list { |
| 7 | unsigned int inner\_ip; |
| 8 | unsigned int inner\_port; |
| 9 | unsigned int outside\_ip; |
| 10 | unsigned int outside\_port; |
| 11 | unsigned int protocol; |
| 12 | unsigned int id; |
| 13 | unsigned long long time; |
| 14 | struct socket \*sock; |
| 15 | struct nat\_connect\_list \*next; |
| 16 | struct nat\_connect\_list \*last; |
| 17 | }NAT\_connect\_list; |

NAT规则在NAT表中定义，其中：

NAT\_list结构存储静态NAT映射，每条记录包括inner\_ip（局域网IP）和outside\_ip（广域网IP）。

NAT\_connect\_list结构维护动态NAT连接状态，包括内部和外部IP、端口信息，以及协议类型。它的字段设计为：

inner\_ip 和 outside\_ip：内网和外网的IP地址。

inner\_port 和 outside\_port：内网和外网的端口，结合IP地址实现端口转换。

id：用于记录ICMP报文的id。

time：用于实现超时后自动删除。

protocol：指明所支持的协议。

sock：记录套接字信息，用于数据转发。

next 和 last：用于连接链表的双向指针

## 模块接口设计

### 防火墙过滤接口

my\_firewall.c 中的 telnet\_filter 函数是防火墙模块的过滤接口，负责匹配入站和出站的网络数据包。该接口利用 Netfilter 钩子函数注册，在数据包到达网络层时被触发。过滤接口根据规则表和连接表实现包的放行或丢弃。

**流程：**

先调用 connect\_match 检查当前数据包是否已在连接表中（已获许可的连接），如果存在则直接放行。

若不在连接表中，则调用 rule\_match 匹配规则表。规则匹配成功则放行，并在连接表中记录该条目；否则丢弃数据包 fw\_in钩子用于入站数据包检查，fw\_out钩子用于出站数据包检查，分别在 NF\_INET\_LOCAL\_IN 和 NF\_INET\_LOCAL\_OUT 阶段注册 。

### NAT处理接口

telnet\_NAT\_out和telnet\_NAT\_in 是出站和入站 NAT 转换的实现接口，用于管理数据包的 NAT 处理。

**流程：**

telnet\_NAT\_out：将私有 IP 转换为公共 IP 地址。根据数据包的源 IP、端口等信息匹配 NAT 表，找到匹配条目后替换源 IP 和端口为公共 IP 和 NAT 端口。若无匹配，对于符合规则的IP则创建新 NAT 条目 。

telnet\_NAT\_in：将公共 IP地址转换为私有IP地址，实现 NAT 的逆向转换，通过连接表匹配找到对应的内部 IP 和端口。

**接口注册：**

NAT 出站和入站接口分别注册在NF\_INET\_POST\_ROUTING和NF\_INET\_PRE\_ROUTING阶段，确保数据包在路由前后都可以进行 NAT 处理。

### Netlink 通信接口

Netlink 接口用于用户态与内户态的 UI.c 和 log.c 文件与内核态的 Netlink 接口交互，实现规则更新、日志传输等功能。

**Netlink 初始化：**

netlink\_init() 创建一个 Netlink socket，绑定到特定的 Netlink 协议类型。

**Netlink 发送和接收：**

netlink\_send 函数用于从用户态向内核发送数据。

nltest\_krecv 函数是内核态的接收函数，处理用户态发来的指令。

### 日志记录接口

print\_log 函数用于生成并发送日志信息，便于审计和故障排查。日志包含源 IP、目的 IP、协议类型和匹配规则状态等信息。

日志记录发生在规则匹配时（rule\_match）和连接检测时（connect\_match），根据匹配结果调用 print\_log 并通过 Netlink 发送到用户空间的log程序。

### 校验和更新接口

数据包的 NAT 转换会改变 IP 和端口，因此需要重新计算校验和。update\_checksum 函数根据不同协议（TCP/UDP）的校验和要求分别调用update\_tcp\_checksum和update\_udp\_checksum。

作用是确保经过 NAT 转换的数据包符合网络协议的校验和要求，提高传输的可靠性。

# 问题分析与解决

## 内核卸载后卡死问题

### 问题描述

在使用rmmod命令卸载内核后，系统直接卡死。

### 原因分析

在使用中不会发生卡死，只要卸载内核就会卡死，说明错误出在module\_exit()调用的函数中。对应的myfw\_exit函数中查找错误。在myfw\_exit中是释放资源的操作，仔细检查发现，在卸载netfliter的挂钩时少卸载了一个，这个和用户空间中close文件不太一样，如果没有close文件系统会自动close，不会导致出错。

### 解决方法

补齐卸载挂钩的函数即可。

## 内核卸载后卡死问题

### 问题描述

在使用防火墙更新规则表之后，系统直接卡死。

### 原因分析

根据出现问题的时间节点，每次都是卡在更新规则表后的一瞬间，所以从内核模块接收规则表后开始查找错误。发现唯一可能导致错误的部分是，在接收到规则表后，系统释放掉所有的连接时调用的kfree函数导致的崩溃。仔细检查发现，时kfree函数在释放空间前没有检查该指针是否为空指针，导致释放到了NULL空间导致的崩溃。

### 解决方法

在kfree函数前添加空指针检查操作。

## 内核卸载后卡死问题

### 问题描述

经过 NAT 转换的数据包在接收端被丢弃，wireshark抓包发现没有对应的数据包。

### 原因分析

可能是在NAT实现函数里面，错把ACCEPT写成DROP了，但是仔细检查后发现没有，后来询问同学得知，NAT修改了数据包头后需要修改校验和，否则会被设备自身的防火墙丢弃。

### 解决方法

添加IP头和TCP/UDP头的校验和的重计算函数。

## 使用NAT后UDP接收报文就退出问题

### 问题描述

经过 NAT 转换后Docker向Windows11主机发送报文正常，但是主机向Docker回复后立马就显示连接中断，但实际上Docker收到了NAT转换后的报文。

### 原因分析

经过Wireshark抓包发现，主机向Docker回复报文后，会收到一个ICMP的端口不可达消息，这让我想起来，我在进行NAT转换后，新的端口是随机产生的，并没有将端口注册到Socket上，导致收到报文后，这个端口实际上是没有打开的，Linux内核会认为这个回复报文是一个错误，从而回复ICMP的端口不可达。

### 解决方法

在NAT使用了WAN中的一个端口后，利用Socket注册这个端口即可。

# 系统测试

## 测试环境

网络拓扑：存在一个Ubuntu20.04虚拟机，采用网卡桥接方式与Windows11真机连接，同时Ubuntu20.04虚拟机中存在一个Docker（用于测试NAT功能），与虚拟机之间不在同一个网段。详细网络拓扑结构见图7-1。

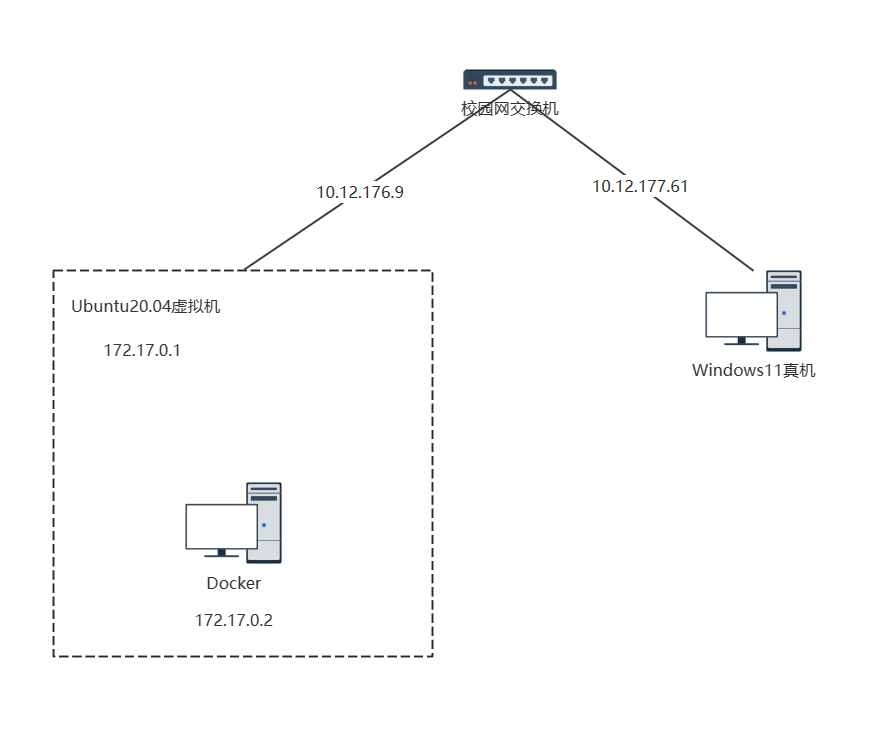


图7-1实验环境网络拓扑图

## 功能测试

### UDP拦截测试

首先，添加防火墙规则放行所有报文，见图7-2。

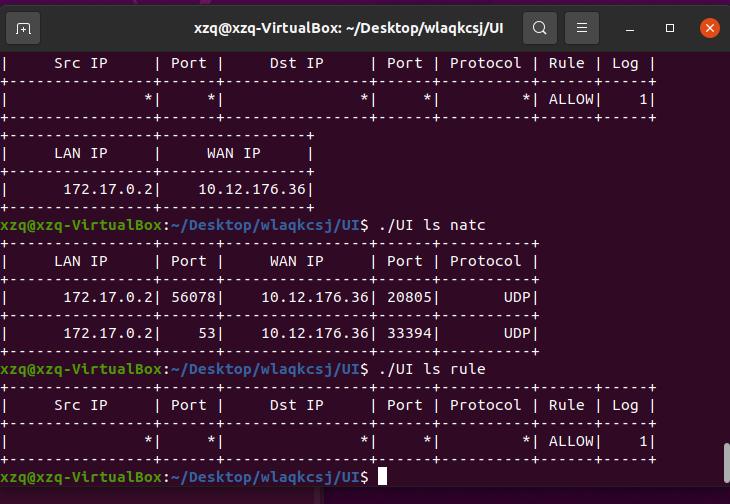


图7-2 添加放行所有报文的规则

然后测试UDP功能，图7-3为Ubuntu20.04虚拟机内发送UDP数据包，图7-4为Windows11真机接收到数据包。



图7-3 Ubuntu虚拟机开启UDP客户端

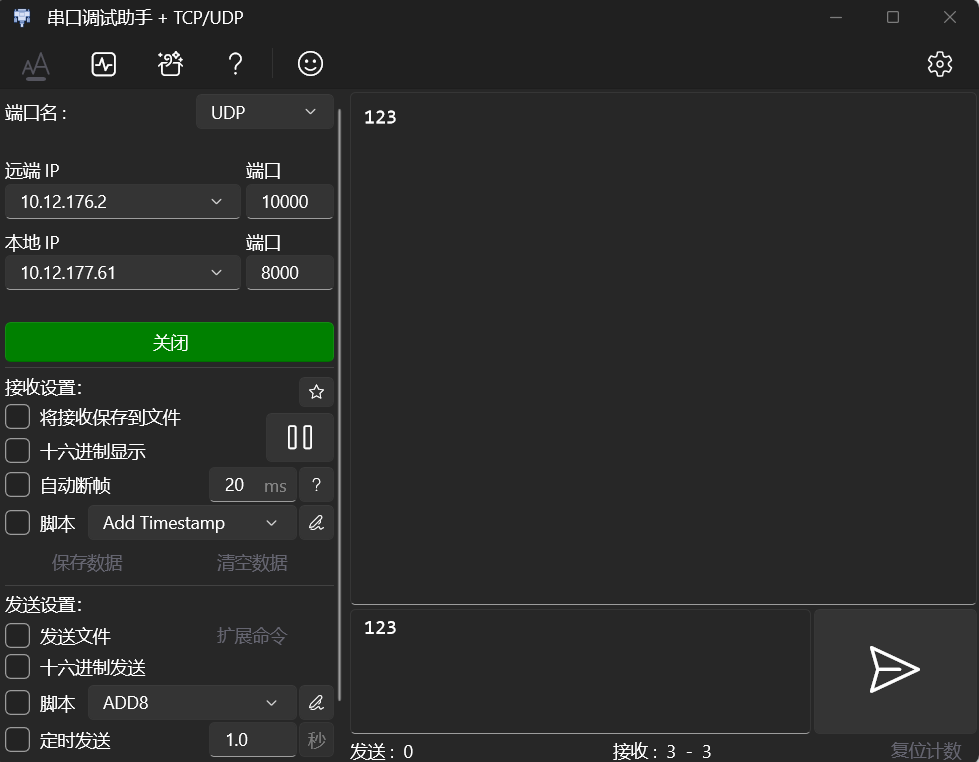


图7-4 Windows11主机开启UDP服务端

此时修改防火墙规则，见图7-5。拒绝从10.12.176.2:10000到10.12.177.61:8000的UDP报文。

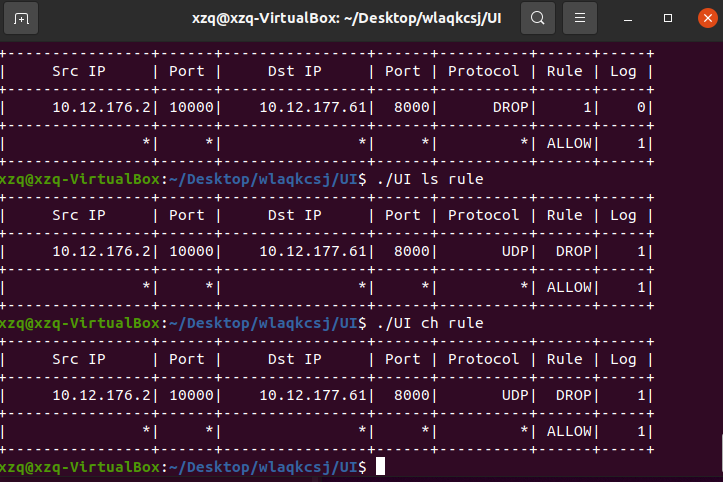


图7-5 添加拒绝规则

可以看到，内核中拦截了这个报文，在内核中输出了拦截信息，见图7-6。真机也接收不到报文。

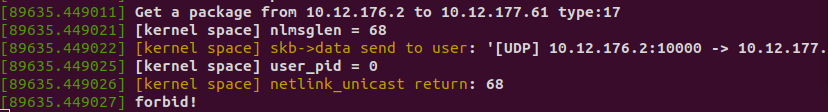


图7-6 内核拒绝报文输出

但是，如果我们从真机向虚拟机发送报文，则可以接收，内核输出了放行的信息，见图7-7。实际上虚拟机确实能收到报文

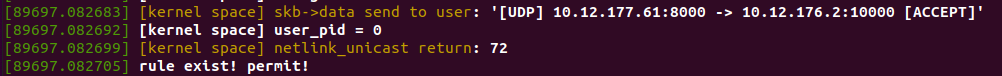


图7-7 内核放行从真机发来的报文

综上，说明UDP防火墙功能正常。

### TCP拦截测试

方案和UDP拦截测试类似，正常通信状态不再赘述，直接添加拦截的规则拒绝从10.12.176.2:60812到10.12.177.61:8000的TCP报文，见图7-8。

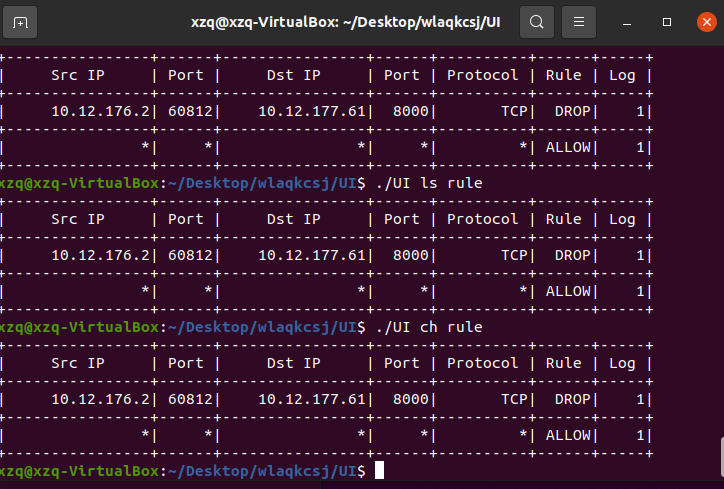


图7-8 添加拒绝规则

此时，Windows11真机开始作为服务端监听TCP连接，由于TCP建立连接的过程需要三次握手，所以需要双方能互发报文，所以我们的测试是先建立起连接，再添加防火墙规则，拦截从虚拟机到真机的防火墙内核输出见图7-9。从效果上看虚拟机内的TCP客户端确实无法发出报文，由于收不到ACK不停的重传。

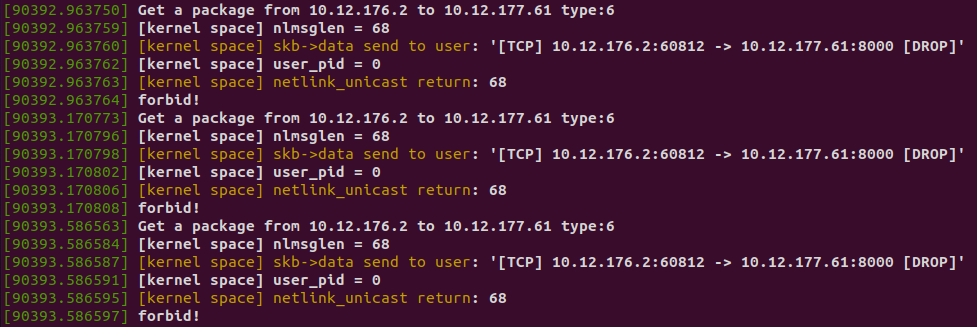


图7-9 虚拟机内发给服务端的TCP被拦截

但是从真机发给虚拟机的报文并不会被拦截，测试效果见图7-10。

****

图7-10 虚拟机收到来自真机的报文

内核输出放行信息见图7-11。

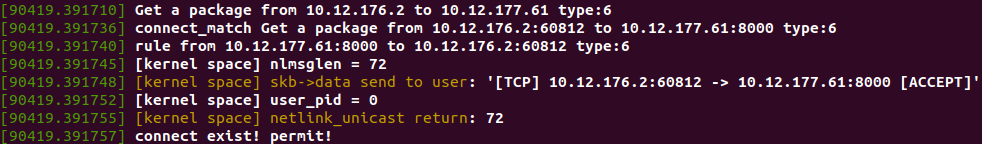


图7-11 内核放行来自真机的报文信息

综上所示，说明TCP拦截功能正常。

### ICMP拦截测试

由于ICMP比TCP和UDP都要简单，效果类似，拦截原理相同，此处不再赘述。具体效果可看演示视频。

### 状态检测测试

我们建立一个TCP连接，可以看到，在第一个报文是判断的规则表，见图7-12。

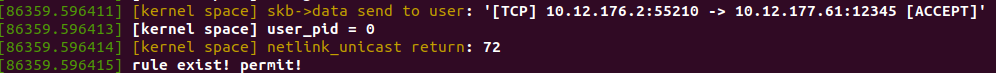


图7-12 报文放行是因为规则表

之后的数据交换检测的都是连接表。

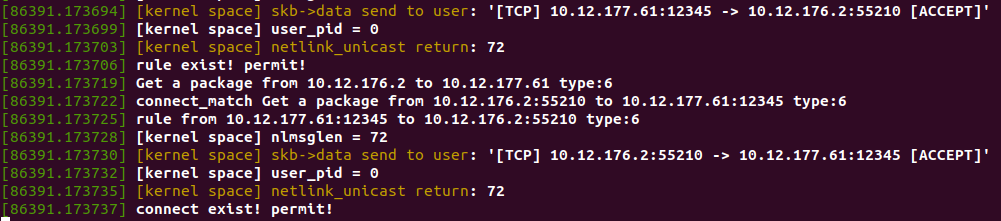


图7-13 报文放行是因为连接表

可以查询目前建立的连接表，见图7-14。可以看到第二行就是我们建立的连接双方的IP和端口。

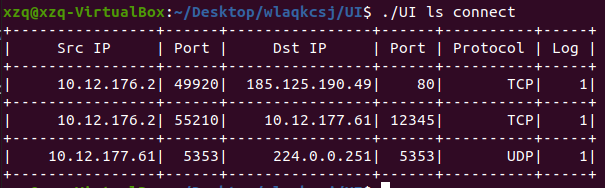


图7-14 查询连接状态

### NAT功能测试

在Docker中使用hping3测试UDP连接，在主机上编写Python脚本，可以监听端口的UDP信息，并且自动回复相应信息。Python脚本见图7-15。



图7-15 真机上监听UDP的脚本

当我们关闭NAT功能时，hping3收不到回应报文，因为主机并不知道172.17.0.2是谁。见图7-16。

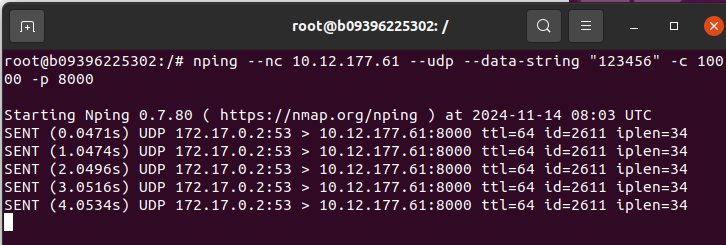


图7-16 虚拟机接收不到返回的UDP报文

当我们打开NAT功能后，可以看到，hping3收到了回复报文，说明此时IP已经被转换了。hping3效果见图7-17。

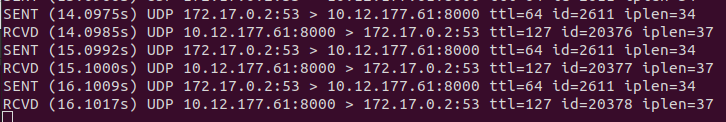


图7-17 虚拟机接收到了返回报文

成功转化IP后，在Python脚本中，可以看到回复IP已经变为了10.12.176.2（Ubuntu20.04虚拟机IP），见图7-18。证明NAT转换成功。

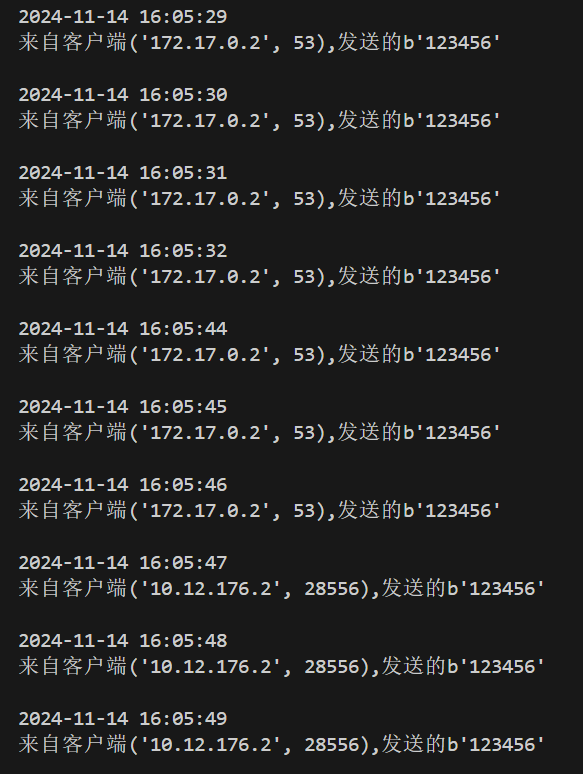


图7-18 在真机视角源IP已经发送变化

此时，我们可以查看NAT连接表，在系统中已经建立了一个172.17.0.2:53->10.12.176.2:33699的NAT转换连接（端口变化的原因是两次截图时间间隔太久，中途我关闭了hping3，导致再次开启后动态分配的端口变化）。见图7-19。

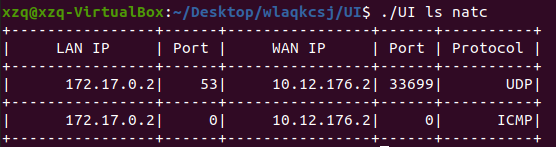


图7-19 NAT的连接表

## 性能测试

经过ping命令测试，开启了防火墙和未开启防火墙的响应时间几乎没有变化，说明性能良好，不会造成系统网络拥堵。

同时，操作中系统流畅，不会出现卡顿等影响体验的现象。

# 心得体会及意见建议

## 心得体会

在本次实验中，开发 NAT 模块和防火墙过滤功能不仅是一次技术实践的历练，更是对网络安全和编程规范化的重要性有了更深层次的理解。防火墙和 NAT 模块的核心功能在于保护内部网络的安全性和稳定性，在实现这些功能的过程中，涉及到了大量链表操作、内存管理、定时清理机制等内容，这些操作一方面为系统提供了高效的网络安全防护，另一方面也对代码质量提出了严格的要求。尤其是在防火墙规则匹配和 NAT 连接管理中，我发现了多个常见编程错误的可能性：如内存泄漏、链表操作错误、校验和更新失败等，这些问题如果不加以规避，不仅会影响防火墙的性能，还可能导致系统的不稳定。特别是在链表操作和内存管理方面，一些小的细节往往会导致严重的后果。例如，由于 NAT 模块和防火墙模块的规则表需要频繁增删条目，如果链表操作中忽略了节点指针的正确维护，很容易出现段错误或者内存泄漏，从而导致系统资源被不必要地占用，最终影响系统的运行效率和稳定性。因此，开发中不仅需要对规则匹配和 NAT 转换逻辑保持清晰，更要在代码实现时注重每一个细节，保证代码的健壮性和可维护性。这让我深刻体会到，在编写任何功能代码时都必须关注细节，培养良好的编程习惯和高度的代码规范意识。

NAT 模块的实现涉及对数据包进行出站和入站转换，将私有 IP 地址和端口映射为公共地址和端口，从而实现内部网络与外部网络的安全通信。由于 NAT 转换会改变数据包的源 IP 和目的 IP，在这之后还需要重新计算校验和以保证数据包的完整性，这一过程看似简单，但在不同的协议（如 TCP、UDP 和 ICMP）下，校验和的更新逻辑有所不同，因此在每次进行 IP 或端口转换后都需要分别更新不同的校验和。实验初期我未充分注意到这一点，导致在接收端接收的数据包被丢弃，经过多次调试才发现是校验和更新错误造成的。这个经历也提醒我在编写代码时要全面考虑各种可能性，理解每一步操作的原因和后果。经过调试和改进，我逐渐掌握了校验和更新的机制及其在 NAT 转换中的重要作用。除此之外，实验中还涉及到使用随机端口生成机制来确保 NAT 出站数据包的端口唯一性。在高并发的网络环境中，端口冲突可能会导致不同连接的数据包被混淆，这时便需要使用合理的随机生成策略，并确保生成的端口不会与现有的 NAT 表产生冲突，保证每一个 NAT 映射的唯一性。这让我认识到，网络安全中的任何一项技术实现都离不开对细节的严格控制，安全本质上是通过对细微之处的掌控来实现的。

此次实验，我对网络安全的意义有了更深刻的体会。防火墙和 NAT 模块的设计不仅仅是实现某个技术需求，更是在网络传输的各个环节构筑安全屏障。在当今信息化高速发展的背景下，社会中的许多关键基础设施都依赖于网络进行信息交换，网络安全问题不仅关乎个人数据安全，更影响着社会的稳定运行。防火墙和 NAT 模块作为网络安全的基础功能，承担着防止外部攻击、保护内部数据的重任，保证了内部网络环境的有序运行。这种保护不仅仅是对单一网络系统的防护，更是在无形中为信息社会提供安全保障。对于我来说，开发这样一项系统防护功能不仅是技术上的挑战，也是一次责任感的提升。网络安全不仅是一项专业技术，背后更承载了对国家和社会的信息保障责任。作为一名未来的网络技术人员，我深知需要通过不断提升技术水平来肩负起保护网络安全的责任。在未来的学习与工作中，我将时刻铭记这一使命，以更加严谨和认真的态度去面对每一次技术实践，不断提升自己，为构建更为安全、稳定的网络环境贡献自己的力量。

## 意见建议

实验中如果我们需要实现过滤指定报文，需要使用相应工具构造报文，如果能提供相应的工具构造出我们需要用于测试的网络报文可以更方便我们进行调试。