硕士学位论文

基于 DPDK 的高性能 VPN 网关的研究与实现

RESEARCH AND IMPLEMENTATION OF HIGH PERFORMANCE VPN GATEWAY BASED ON DPDK

穆瑞超

哈尔滨工业大学

2017 年 6 月

国内图书分类号：TP393　　学校代码：10213
国际图书分类号：004.7　　密级：公开

工学硕士学位论文

基于 DPDK 的高性能 VPN 网关的研究与实现

硕 士 研 究 生：穆瑞超
导　　师：佟晓筠教授
申 请 学 位：工学硕士
学　　科：计算机科学与技术
所 在 单 位：计算机科学与技术学院
答 辩 日 期：2017 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index: TP393
U.D.C: 004.7

Dissertation for the Master Degree in Engineering

RESEARCH AND IMPLEMENTATION OF HIGH PERFORMANCE VPN GATEWAY BASED ON DPDK

Candidate: Mu Ruichao
Supervisor: Prof. Tong Xiaojun
Academic Degree Applied for: Master of Engineering
Speciality: Computer Science and Technology
Affiliation: School of Computer Science and Technology
Date of Defence: June, 2017
Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

较之于专用的虚拟专用网（Virtual Private Network，VPN）网关设备，在通用服务器上部署软件 VPN 网关成本低、灵活性强、对新兴技术适应性强，提升其性能有利于其应对逐渐增大的 VPN 通信流量，有较大实际意义。本文致力于提升软件 VPN 网关吞吐量、包转发率和时延等性能。

首先，为解决通用服务器内核网络处理低效的问题，本文研究了数据面开发工具集（Data Plane Development Kit，DPDK）和用户态协议栈应用在 VPN 网关中的方法，对比传统软件 VPN 网关设计了基于 DPDK 的 VPN 网关框架。该框架使用用户态驱动，利用轮询模式收发报文，构建精简的用户态协议栈，并在用户态实现 VPN 网关的连接转发功能和代理转发功能。

为在用户态实现连接转发功能，本文优化了基于 Patricia 树的路由算法，删去了其中对掩码的操作，在查找过程中找到叶子结点即可，不再回溯，更加适用于 VPN 路由查找。实验表明，该算法在少量用户分散于大量网络地址的条件下灵活性强、查找速度快。在此基础上，结合 VPN 路由查找的特点，提出一种基于划分阶段的改进方法，该方法将 VPN 路由查找分为两个阶段，第一阶段使用 Patricia 树查找网络地址，第二阶段使用 hash 表确定具体虚拟 IP。实验表明，该改进方法在大量用户集中于少量网络地址的条件下效率较高。

为在用户态实现代理转发功能，本文研究了用户态网络地址转换（Network Address Translation，NAT），利用映射描述了 NAT 技术的核心，并给出一种 NAT 核心算法。在核心算法的基础上，本文设计了一种用户态 NAT 实现方法，该方法使用两个 hash 表记录 NAT 规则，实现了流出报文的源地址转换，流入报文的目的地址转换以及 NAT 规则的超时删除。通过实验，本文为该实现方法选了 BPHash 作为其 hash 函数。通过测试，验证了该实现方法能够完成用户态 NAT 功能。

最后，在上述研究基础之上，本文设计并实现了基于 DPDK 的软件 VPN 网关，并与其他五种软件 VPN 网关进行了对比测试。测试结果表明，本文实现的软件 VPN 网关的系统吞吐量、包转发率、传输时延等性能均优于其他五种网关，并且包长越小优势越明显。

关键词：软件 VPN 网关；DPDK；用户态协议栈；VPN 路由查找；用户态 NAT

Abstract

Comparing with dedicated Virtual Private Network (VPN) gateway device, software VPN gateway deployed on a generic server has an advantage in price, flexibility and adaptability to new technologies. Therefore, it has great significance to improve its perforemance to deal with increasing VPN traffic. This paper dedicates to improve performance of software VPN gateway by increasing its throughput, packet forwarding rate and reducing its latency.

First of all, to solve inefficent network processing problem of generic server kernel, this paper studies the application of Data Plane Development Kit (DPDK) and userspace protocol stack in VPN gateway. Comparing with the traditional method, this paper designs a DPDK-based framework for software VPN gateway which implements userspace driver, uses poll mode to receive and send packets, creates its custom userspace stack and implements connection forwarding function and proxy forwarding function in userspace.

To implement connection forwarding function in userspace, this paper optimizes the Patricia tree based routing lookup algorithm by removing the opration for mask so that the lookup ends when the leaf is found and does not need backtracing . The optimized algorithm is more suitable for VPN routing lookup. The expriments show that the algorithm is flexible and faster under a condition that a few users disperse to many network segments. Then, this paper proposes a improved method for this algorithm based on the features of VPN gateway. The method is dividing the searching into two stages. The first stage is to find network segment by using the Patricia tree. The second stage is to find the target virtual IP by using a hash table. The expriments show the method has a higher performance under a condition that a lot users concentrate in a few network segments.

To implement proxy forwarding function in userspace，this paper studies userspace network address translation (NAT). This paper describes the core of the NAT by mapping and proposes a NAT core algorithm. On the basis of the algorithm, this paper designs a userspace NAT implementation which records the NAT rules by two hash tables, implements source address translation for outgoing packets, implements destination address translation for ingoing packets and implements timeout removing for NAT rules. The BPHash is chosen as its hash function by an expriment. Then, a test is made and verifies that the design can implement the function of userspace NAT.

Finally, on the basis of the researches above, this paper designs and implements a software VPN gateway based on DPDK and tests it with other five software VPN

gateways. The test results show that the performance of software VPN gateway based on DPDK is superior to other five software VPN gateways in system thoughput, packet forwarding rate and transmission latency, especially under the circumstance of short packet payload.

Keywords: software VPN gateway, DPDK, userspace protocol stack, VPN routing lookup, userspace NAT

目 录

第1章 绪 论

本章主要阐述课题研究背景和意义，介绍与分析国内外对于 VPN 网关加速问题的研究现状，并给出本文的研究内容与组织结构。

1.1 研究背景和意义

虚拟专用网络（Virtual Private Network，VPN），使用隧道技术和加密技术在公用网络中建立虚拟专用链路，使地理上相隔较远的 VPN 用户也能够像在局域网中一样通信。相较于专用链路，VPN 成本低，灵活性强，常被用于跨地域企业的内网互联。随着 VPN 技术的发展，VPN 被广泛应用其他领域，例如网络虚拟化领域、云计算的接入领域、网络安全领域[1]和移动安全领域[2-4]。

在各类 VPN 应用中，VPN 网关作为星形 VPN 网络的核心，在整个 VPN 网络中起到关键作用。部署 VPN 网关可以采用专用 VPN 设备，也可以通过在通用服务器上部署软件 VPN 网关的方式实现。相较于专用设备，软件 VPN 网关具有以下优点。第一，软件 VPN 网关成本较低，能够充分利用现有服务器资源，不必采购专用设备。第二，软件 VPN 网关灵活性强。一方面，软件 VPN 网关的部署不依赖专用设备，当不再使用 VPN 功能时，服务器可以做其他用途，不会造成闲置。另一方面，软件 VPN 网关功能增减相对容易，能灵活应对需求变更。第三，软件 VPN 网关对新兴领域适应性强。在网络虚拟化和云计算的浪潮下，VPN 作为一种基本的网络技术和思路，在新兴领域有广泛的应用。软件 VPN 网关更符合这些新兴领域对低成本和高灵活性的要求。

然而，传统软件 VPN 网关吞吐量、包转发率、时延等性能较差。随着网络技术的发展，VPN 通信规模越来越大，用户对响应时间的要求越来越高，软件 VPN 网关面临越来越大的性能压力。因此，研究软件 VPN 网关加速技术有十分重要的意义。

1.2 国内外研究现状

1.2.1 传统 VPN 网关加速的研究现状

传统 VPN 网关加速的研究主要围绕 VPN 的功能优化展开，包括安全协议

的改进、加密效率的提升和网关架构的改进等。

（1）安全协议的改进方面

在 VPN 协议研究早期，Shue 等人[5]使用开源的 IPSec 代码 Openswan，通过记录每种操作需要的时间，评估了 IPSec 性能瓶颈，该文献指出连接的建立和保持阶段比通信阶段消耗大，更容易成为瓶颈，加密占执行时间的 32%到 60%。之后，他们评估了 IPSec VPN 服务器的性能，指出密钥交换开销较大，并提出了一种加密安全缓存恢复协议来降低开销[6]。单蓉胜等人[7]抽象出了安全网关的原子操作集和操作序列表达式，改进了安全网关的访问控制表，提出一种安全快速的处理算法，并通过实验验证了新的算法和软件结构能够提升安全网关的吞吐率，降低分组处理时延，并保证策略数目不会影响吞吐率的变化。2010 年，Narayan 等人[8]研究了 Windows 操作系统不同版本对 VPN 性能的影响，发现 Windows 操作系统版本对 VPN 协议性能影响较小。而在 2015 年，Padhiar 等人[9]比较了多种操作系统对常见 VPN 协议性能的影响，指出 VPN 协议、操作系统都会影响 VPN 隧道的网络性能。2016 年，文献[10]使用开源工具 Quagga 和 strongSwan 实现了一种 VPN 软件路由器，并测试和分析了该软件路由器支持的加密和 hash 算法，为配置最优的 VPN 参数提供了参考。文献[11]分析了 VPN 使用 SSTP 和 IKEv2 协议时的吞吐量、抖动、时延等参数，指出 IKEv2 较之于 SSTP 性能占优势。

（2）加密效率的提升方面

在加密效率研究早期，梅松等人[12]在对 VPN 体系进行剖析的基础上，提出了一种基于流水线和多卡并行处理的模型，并分别对流水线模型和多卡并行处理模型进行了分析，推导出了系统性能和加密卡的速度、并行加密卡的个数以及性能损失率之间的数学关系。2014 年，李湘峰等人[13]研究了对称密钥加密算法在 IPSec 协议中的应用，分析了 DES、AES、SM4 等多种加密算法的实现特点、复杂度和安全性，比较了算法所需资源和实现速度，为 IPSec 应用提供了建议。2016 年，Turan 等人[14]为缓解 VPN 应用中加密对 CPU 的压力，设计了一种协处理器，专用于加速网络加密库，并将该设计应用于开源代码 SigmaVPN 中。测试表明，对大小为 1024 字节的帧，该设计能够减少 93%的加密执行时间，并使得 TCP 和 UDP 带宽提升 4 到 5 倍。

（3）网关架构的改进方面

在提升 VPN 执行效率研究早期，文献[15]提出了基于多核处理器的 IPSec 协议并行处理模型，将 IPSec 报文分流到多个 CPU 核中，提高传输带宽。测试

表明，在 Linux 双核处理器上，该方法能够使 IPSec VPN 网关性能提升近一倍。2012 年，周振斌等人[16]将负载均衡策略引入 IPSec VPN 网关。在网关处组建服务器集群，利用特定的负载均衡策略，实现了对 IPSec 数据流的均匀分配。并通过模拟实验验证了负载均衡技术可以一定程度减少 IPsec VPN 系统响应时间，提高系统的吞吐量。2015 年，文献[17]指出 VPN 协议的处理是高速网络实现的瓶颈，介绍了一种基于 FPGA 的万兆以太网 IPSec ESP 协议栈的设计，支持隧道模式和传输模式，具有抗重放能力，通过采用多级流水操作、多缓存乒乓操作、多进程并行处理等技术实现了万兆线速。

1.2.2 网络处理加速的研究现状

近些年来，随着网络技术的发展，网络处理对系统的要求越来越高，通用服务器已经不能跟上高速网络接口的步伐，涌现出很多专用的网络服务器。但专用的网络服务器价格昂贵，灵活性差，很多时候不能满足实际需求。对于通用服务器网络处理的瓶颈，文献[18]指出通用性和性能是矛盾的，服务的通用性越好，在特殊领域的高性能就越容易遇到瓶颈，为通用性设计的操作系统的网络处理效率不高。文献[19]指出通用操作系统的通用 IP 协议栈较之于其硬件性能较低，更容易遇到瓶颈，导致硬件利用率不高，降低了系统整体性能。文献[20-22]也都指出通用服务器内核的网络处理功能存在瓶颈。

针对通用服务器内核网络处理功能的低效，研究人员提出了很多高性能软件报文处理框架。在这些平台中，认可度较高、应用较广的[23]是 DPDK[24]、netmap[25]、PF_RING ZC[26]。在这些平台基础上，有很多高性能报文处理应用的研究。将这些研究按框架分为 DPDK、netmap、PF_RING ZC 框架的研究和其他框架的研究，这些研究包括以下内容：

（1）DPDK、netmap、PF_RING ZC 框架的研究

2014 年，文献[19]在 DPDK 平台上实现了一种轻量级 TCP/IP 协议栈，称为 LwIP 协议栈，精简了 TCP/IP 协议栈，并为用户提供使用接口，为上层应用提供高性能包处理服务。2016 年，文献[27]基于 DPDK 平台和网卡的处理器任务卸载技术在通用服务器上实现了一种 IPSec VPN 网关 MoonGen IPSec，单核能达到 10GbE 线速。但该文中提到的网卡加速技术实际是一种硬件技术，不适用于所有网卡，并且该文仅针对 IPSec，不适用于所有 VPN 网关。Ajayan 等人使用 DPDK 在 x86 通用硬件上实现了一种高性能的报文生成系统 Hiper-Ping[28]。该系统用于其他高性能网络应用的性能测试，有较高的性能和较好的稳定性。

PacketUsher[29]是一个基于 DPDK 的高性能报文 IO 引擎，能够提升 I/O 密集型应用和计算密集型应用的性能。其中对 I/O 密集型应用性能的提升达到了专用商业设备的程度，对计算密集型应用性能的提升达到了原来的 3 倍。文献[30]基于 DPDK 开发了一种高效的报文处理框架 OpenNetVM。它通过动态创建服务链来提升路由效率，能够简化网络功能的开发、管理。何佳伟等人[31]提出了一种基于 DPDK 的网络审计系统的设计方案。该方案采用多核多进程架构，具有较好的稳定性，在性能上比使用 libpcap 的方案有较大提升。2017 年，文献[32]提出一种针对 netmap 平台网络处理的能量节约算法，并做了评估。该算法的核心是计算出当前任务所需的最小核数，让其他核处于低消耗状态。文献[33]将 PF_RING ZC 应用在商用服务器的单向通信中，取得了较高的性能。

（2）其他框架的研究

2010 年，Han 等人提出[34]一种高性能软件路由器框架 PacketShader，使用 GPU 进行加速，转发性能很高，并且可以应用在多种不同的包处理应用中。2014 年，文献[35]为降低新协议的开发难度、增加可扩展性并提升性能，实现了一种用户态 TCP 协议栈 MultiStack。HTTP 服务器使用 MultiStack 的 TCP 性能比使用内核协议栈的性能高出 18%-19%。2015 年，TANG Lu 等人[36]指出，专用的网络处理器能达到较高性能，但也有很多缺点，例如，代价高昂，开发周期长，开发难度大等。他们研究了通用多核处理器的高性能报文处理方法，提出一种通用多核处理器结合 FPGA 的报文处理方法。2016 年，Li 等人提出[37]一种基于 FPGA 的可编程高性能报文处理加速平台 ClickNP，其编程语言类似 C 语言，高度灵活，能够提升十几倍到几十倍吞吐量，降低十几倍到几十倍时延，任何包长下都能达到 40Gbps 线速。2017 年，文献[38]研究了 Intel Xeon Phi 的低层线程机制，为该处理器研制了一种新的处理器任务卸载框架 Knapp，来加速包处理。

1.2.3 国内外研究现状简析

传统 VPN 网关加速的研究主要围绕功能优化展开，包括安全协议的改进、加密效率的提升和网关架构的改进等。近些年来，研究人员发现通用服务器内核在进行高速报文处理时存在瓶颈。针对内核处理低效问题，研究人员设计了很多框架来加速报文处理。这些框架分为两类，一类偏向使用硬件例如 GPU 或 FPGA 等辅助 CPU 进行处理，一类偏向使用软件，绕开内核报文收发和内核协议栈，使用高性能报文收发平台和用户态协议栈加速报文处理。偏向硬件的框

架需要借助于 GPU 或 FPGA，灵活性较差，出现问题不容易调试，应用并不广泛。在偏向软件的框架中，认可度较高，应用较广的是 DPDK、netmap 和 PF_RING ZC。在这三种框架中，DPDK 和 PF_RING ZC 的性能高于 netmap[23]。DPDK 遵守 Open Source BSD 许可证，而 PF_RING ZC 使用 EULA license，需要为每个端口或 MAC 地址申请一个许可证。DPDK 加入了 Linux 基金项目，开发者众多，包括很多著名的公司，而 PF_RING ZC 的开发人员较少，应用也没有 DPDK 广泛。所以本文基于 DPDK 框架进行研究。

高性能报文处理平台结合用户态协议栈框架给软件 VPN 网关加速提供了新的思路。VPN 网关也属于网络处理应用，但将高性能报文收发平台结合用户态协议栈框架应用到 VPN 网关中的研究较少，现有的一些应用有些是为通用软件设计的，有些使用了高性能网卡的处理器任务卸载功能。

1.3 本文研究内容及组织结构

本文研究利用 DPDK 平台提升软件 VPN 网关吞吐量、包转发率和时延等性能的方法。为解决通用服务器内核网络处理低效问题，将研究 DPDK 高性能报文收发平台在 VPN 网关的应用。DPDK 平台采用用户态驱动，并且输出是链路层报文，而 DPDK 平台又不提供上层协议栈，故需要构建精简的用户态协议栈，并在用户态实现 VPN 功能。为在用户态实现 VPN 的连接转发功能和代理转发功能，将研究两项关键技术，一是 VPN 路由查找，二是用户态网络地址转换（Network Address Translation，NAT）。在这些研究内容的基础上，给出原型系统的实现，并和现有软件 VPN 网关进行对比测试。本文的组织结构描述如下。

第 1 章介绍课题的研究背景和意义，总结和分析国内外学者对于 VPN 网关加速问题开展的研究和取得的成果，并对论文工作安排进行规划。

第 2 章将研究如何将 DPDK 平台应用到 VPN 网关中。先介绍 VPN 网关的两种主要功能和评价指标。然后介绍 DPDK 高性能报文收发平台结合用户态协议栈的加速方法。最后，本文将介绍基于 DPDK 的 VPN 网关框架。

第 3 章研究连接转发功能中的 VPN 路由查找问题。优化了基于 Patricia 树的传统路由算法，使之更适合于 VPN 路由查找。在此基础上根据 VPN 网关的特点做出改进，提出基于划分阶段的改进方法。实验表明，在大量用户集中于少量网络地址的情况下，基于划分阶段的改进方法效率较高；在少量用户分散于大量网络地址的情况下，基于 Patricia 树的方法灵活性强，查找速度更快。

第 4 章研究代理转发功能中的用户态 NAT 问题。先介绍 NAT 技术，然后

使用映射描述 NAT 核心技术，给出 NAT 核心算法。在核心算法基础上，设计并实现用户态 NAT，并通过实验为该设计选择 BPHash 作为 hash 函数，最后通过测试验证了该设计能够实现用户态 NAT 功能。

第 5 章在前述内容的基础上介绍基于 DPDK 的软件 VPN 网关原型系统的设计和实现，并和其他五种软件 VPN 网关进行对比测试。测试结果表明，本文实现的基于 DPDK 的 VPN 网关性能优于其他五种网关，并且包长越小优势越明显。

第 2 章 基于 DPDK 的高性能 VPN 网关框架的研究

本章介绍基于数据面开发工具集（Data Plane Development Kit，DPDK）的高性能 VPN 网关框架。首先，介绍 VPN 网关通信原理、两种主要功能和评价指标；然后，总结 DPDK 加速技术，介绍用户态协议栈；最后对比传统 VPN 网关框架说明 DPDK 高性能报文处理平台结合用户态协议栈的软件 VPN 网关框架。

2.1 VPN 网关概述

2.1.1 原理分析

虚拟专用网（Virtual Private Network，VPN）的精髓在于虚拟和专用。“虚拟”指 VPN 通信链路并非真正的专用链路，而是在公用网络中虚拟的专用链路，VPN 网络内部使用虚拟 IP 通信；“专用”指 VPN 通过隧道、加密等技术建立了类似于专用链路的通信信道，屏蔽了底层的实现细节，在这个“专用”网络中，VPN 客户端之间能够像使用专用链路一样通信。

VPN 采用隧道技术达到虚拟专用的目的。隧道技术的实质是报文的再次封装。隧道的发送端把原始的使用虚拟 IP 的报文作为负载，使用加密、签名等手段保证其信息的完整和安全，然后在外层对该负载再次进行封装，这次封装采用的是实际链路中的网络协议，使报文能够在实际网络中进行传输。隧道的接收端使用实际链路的网络协议脱去报文的外层封装，取得负载，将负载解密，得到原始报文，再次对原始报文解析，最后得到发送端发送的信息。

在星形 VPN 网络中，VPN 客户端分别使用隧道技术与 VPN 网关建立连接。VPN 网关在 VPN 通信中起中转作用。在得到发送端原始报文后，VPN 网关对原始报文的目的地址进行解析，根据目的地址不同，将 VPN 功能分为两种，第一种，如果目的地址是 VPN 内网的虚拟地址，对应某台 VPN 内网设备，则将原始报文使用目的设备的密钥进行加密，封装后向目的设备发出；第二种，如果目的地址是外部网络，则在规则允许的情况下，进行网络地址转换（Network Address Translation，NAT），并发送 NAT 后的报文。其中，第二种功能和普通的 VPN 功能即第一种功能有一些区别，它在转发后的部分并没有使用隧道技术，但在客户端和 VPN 网关通信时，使用了隧道技术，符合“虚拟”和“专用”

的意义，因此，本文把第二种功能也归为 VPN 功能。

在后面的章节中，主要讨论在用户态完成上述两种 VPN 功能，其中，第一种功能实现了两个 VPN 客户端之间通信，称为连接转发功能；第二种功能实现了 VPN 客户端通过 VPN 网关访问外部网络，称为代理转发功能。

2.1.2 评价指标

VPN 网关加速的研究中，通常将不同包长下的 UDP 吞吐量、UDP 包转发率、TCP 吞吐量、时延和时延抖动作为 VPN 网关的性能评价指标，例如文献[39]、文献[40]，其他评价指标还有并发连接数等。本文将 DPDK 平台应用在 VPN 网关，主要目标是提升吞吐量和降低时延，故采用 UDP 吞吐量、TCP 吞吐量、包转发率、时延和时延抖动作为评价指标，而暂不考虑并发连接数。将这些评价指标按照 UDP 性能、TCP 性能、时延和时延抖动分为四类，描述如下。

（1）UDP 性能

VPN 网关外层封装通常采用 UDP 协议，故 UDP 性能是一种重要评价指标，包括 UDP 吞吐量和包转发率。吞吐量是指在没有帧丢失的情况下，VPN 网关能够接收并转发的最大速率，单位一般是兆位每秒（Mb/s）。VPN 网关对每个数据报文处理都要消耗设备资源，在资源一定的条件下，吞吐量越高，VPN 网关性能越高。吞吐量小会造成网络瓶颈，影响网络性能。而，故 UDP 吞吐量和 TCP 吞吐量都应作为评价指标。包转发率衡量了设备转发数据包的能力，单位一般是包每秒（pps）。在不同包长下，包转发率不同，通常包长度越小，报文转发率越高。在同一包长下，包转发率越高，表示单位时间内能够处理的报文数量越多，VPN 网关性能越高。

（2）TCP 性能

目前应用多采用 TCP 协议，即 VPN 隧道内层报文 TCP 比例较大，故 TCP 性能也是一种重要评价指标，包括 TCP 吞吐量和 TCP 文件下载速率。其中，TCP 吞吐量是指 VPN 网关能够处理的 VPN 隧道内层 TCP 报文的速率，单位一般是兆位每秒（Mb/s）。TCP 吞吐量越大，VPN 网关性能越高。TCP 文件下载速率也是一种衡量 TCP 性能的指标，是指 VPN 网关的客户端通过 VPN 网关下载文件时的速率，单位一般是兆字节每秒（MB/s）。TCP 文件下载速率越大，VPN 网关性能越高。

（3）时延

时延指 VPN 网关从接收到一个数据包到转发出该数据包所消耗的时间，能

够衡量 VPN 网关的处理速度，单位通常是毫秒（ms）。时延越低，VPN 网关性能越高。时延过高会严重影响用户体验。

（4）时延抖动

时延抖动又称为 jitter，指时延的变化幅度，衡量了 VPN 网关的稳定性。Jitter 值越低表示 VPN 网关稳定性越高。

2.2 DPDK 高性能报文收发平台介绍

2.2.1 DPDK 简介

传统 Linux 系统中，网卡驱动运行在内核态，采用中断方式进行报文收发处理。早期 CPU 工作速度远高于外设访问速度，所以中断处理较为有效，但随着芯片技术和高速网络接口技术的发展，报文吞吐量越来越高，I/O 速率逐渐接近甚至超越 CPU 工作速度，使用轮询技术来处理高速端口更具有优势，这构成了 DPDK 的基础。

DPDK 起初由 Intel 公司提供，为报文高速处理提供驱动和函数库的支持。最初仅支持 Intel 公司的 CPU 和网卡，但它设计的目标不仅仅是支持 Intel 公司产品。目前，CPU 的支持已经扩展到 IBM POWER 和 ARM，网卡的支持已经扩展到大部分主流网卡。Intel 起初只是将 DPDK 以源代码的形式分享给少量用户，但随着行业大幅度接受，2013 年 Intel 将 DPDK 以 BSD 开源方式分享在 Intel 的网站上。2013 年 4 月，6wind 联合其他开发者成立 www.dpdk.org 开源社区。随后，越来越多的个人和企业加入这一社区，使得 DPDK 发展迅速。2017 年 4 月 3 日，DPDK 成为 Linux 基金项目。

2.2.2 DPDK 核心技术

DPDK 可以看作一个开发工具集，或者一个开发平台，它是多种加速技术的融合。本文在实现基于 DPDK 的 VPN 网关时涉及的加速技术主要包括以下几个方面。

（1）网卡性能优化

网卡性能优化主要包括轮询模式、burst 报文收发模式。DPDK 可以采用轮询模式。传统 VPN 网关中，网络数据包的收发由内核完成，采用中断方式，中断方式会带来大量上下文切换消耗。DPDK 的轮询模式，通过循环判断的方式

收发数据包，提升了收发报文的效率。这种方式会使 CPU 始终处于满负荷运行状态，但在需要大量、不间断地处理网络数据包的场景下十分适用。DPDK 支持轮询模式，并不意味着它不支持中断，根据应用场景的需要，中断可以被支持，例如链路层状态发生变化的中断触发与处理。DPDK 可以采用 burst 模式收发报文。Burst 模式是指一次完成多个报文的收发，例如一次发送或接收 8 个、16 个或 32 个报文。DPDK 的 burst 报文收发模式把收发报文的复杂处理过程分解成较小的处理阶段，把相邻的数据访问、相似的数据运算集中处理，通过这种方式，减少对内存或缓存的访问次数，提升收发报文的效率。

（2）用户态驱动

传统 VPN 网关中，报文数据要在内核态和用户态多次切换和拷贝，引起很多不必要的消耗，DPDK 支持用户态驱动，直接在用户态处理网络报文，避免了内存拷贝和系统调用。需要明确的是，使用 DPDK 平台接收到的是链路层报文，DPDK 不提供用户态协议栈，不负责解析接收到的报文，需要 DPDK 使用者自行构建协议栈，对报文进行解析和处理。

（3）Cache 和内存

传统软件 VPN 网关并未充分利用 cache 和内存。DPDK 充分考虑了 cache 和内存来加速报文处理，这些技术包括软件预取、大页内存。

软件预取和硬件预取都利用了程序的空间局部性和时间局部性原理。但软件预取不同于硬件预取，硬件预取机械地按照规则预取数据，不能识别数据访问的规律，在某些情况下不能提高程序执行的效率，而软件预取是在处理器中增加了一些指令，使得软件开发者能够管理部分 cache，将某些热点区域显式地加载到 cache 中，提高执行效率。但如果不能正确使用，软件预取也可能造成 cache 负担过重或无用数据比例增加，反而会降低程序性能。

大页内存能够加速虚拟地址到物理地址的映射。虚拟地址和物理地址的映射采用页表来存储，为了加速映射过程，引入了翻译后备缓冲器（Translation Look-aside Buffer，TLB）cache，用于缓存内存中常用的页表项。在内存一定的情况下，如果页表越小，那么页表项就越多，而 TLB cache 的空间是有限的，页表项越多，TLB cache 不命中的概率就越大。在程序内存占用较多的情况下，大页内存更有优势。DPDK 就利用了大页内存加速了内存处理。

2.2.3 用户态协议栈

用户态协议栈工作在用户空间。早期，出于安全性和用户态进程隔离等方

面的考虑，传统的网络协议栈被嵌入在内核中，称为内核协议栈。内核协议栈能够利用其他内核子系统来获得性能和封装上的便利。二十世纪九十年代，新型网络应用不断涌现，新的网络协议需添加到协议栈中。但是内核协议栈在修改、调试和维护等方面难度较大[41]，于是提出了用户态协议栈的概念。用户态协议栈将全部或部分协议处理功能从内核态转移到用户态完成。用户态协议栈方便了网络设备的定制。近些年，高速网络接口的速率逐渐逼近甚至超越 CPU 的工作速率，内核协议栈已经难以满足高速报文收发的要求，用户态协议栈和高速报文收发平台配合应用在了高速报文处理中，取得了良好的效果。

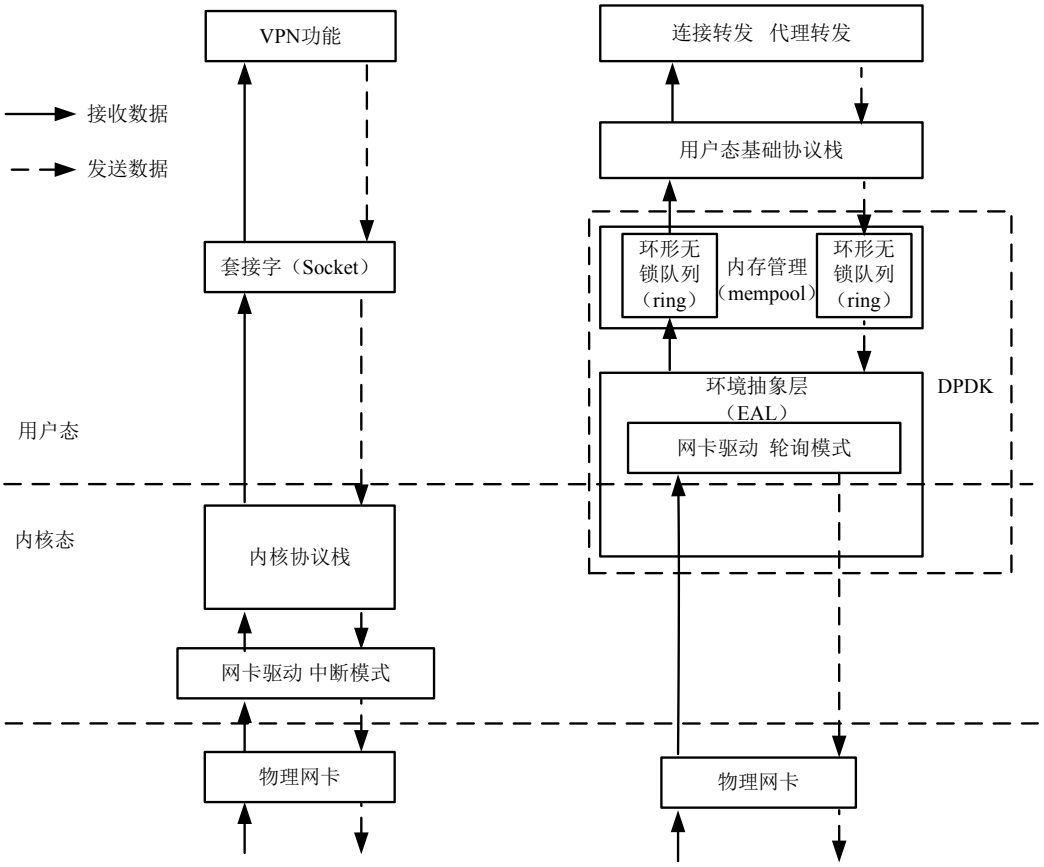
在基于 DPDK 的 VPN 网关中，用户态协议栈的使用有必要性又能够带来性能优势。一方面，DPDK 采用用户态驱动，报文接收后直接在用户态处理，输出是链路层报文，而 DPDK 又不提供用户态协议栈，因此 DPDK 使用者必须构建自己的用户态协议栈来做上层协议处理，这是用户态协议栈的必要性。另一方面，用户态协议栈能带来性能优势，第一，用户态协议栈能够对协议栈进行裁剪，仅保留需要的协议，定制性强，方便根据实际情况进行性能优化；第二，避免了内核态和用户态之间数据交互的拷贝和系统调用。

用户态网络协议栈从狭义上讲，和内核协议栈相对应，就是指内核协议栈转移到用户态的部分，从广义上讲，完成 VPN 功能的协议和设计也可以算做用户态协议栈的一部分。为加以区别，本文将内核协议栈对应的用户态协议栈称为用户态基础协议栈。

2.3 基于 DPDK 的 VPN 网关框架设计

基于 DPDK 的 VPN 网关和传统网关的框架对比如图 2-1 所示。

图 2-1 a)是传统 VPN 网关框架。传统 VPN 网关的网卡驱动工作在内核态，使用中断方式收发报文，并且收发报文需要经过内核协议栈做报文外层封装的解析。VPN 网关进程工作在用户态，通过系统函数接口如 socket 等和内核交换报文数据。有些 VPN 网关在实现时还使用了虚拟网卡，虚拟网卡工作在内核态，用户态进程通过字符设备和虚拟网卡交换数据。虚拟网卡的方式降低了创建 VPN 隧道的复杂性，简化了 VPN 逻辑，方便使用内核中的路由等模块，但引入虚拟网卡会导致数据多次经过内核协议栈，在内核态和用户态多次切换，开销大、效率低。传统 VPN 在实现网络地址转换时，通常采用配置 iptables 规则来实现，iptables 的核心是 netfilter，工作在内核态，也会引起不必要的数据拷贝和系统调用消耗。



a) 传统 VPN 网关框架　　b) 基于 DPDK 的 VPN 网关框架

图 2-1 基于 DPDK 的 VPN 网关与传统 VPN 网关框架对比示意图

图 2-1 b)是基于 DPDK 的 VPN 网关的框架设计。环境抽象层（Environment Abstraction Layer，EAL）实际跨越了核心态和用户态。DPDK 将内存初始化、队列初始化、轮询设备初始化等通用的初始化操作和加速操作都封装在 EAL 中，屏蔽了底层实现细节，简化了编程，使 DPDK 使用者能够专注用户态程序逻辑的构建。用户在初始化 EAL 时，一般仅需调用一个初始化函数即可。

DPDK 使用用户空间驱动，网卡直接和用户态的驱动做数据交换，绕开了内核协议栈，避免了内核态和用户态数据交换的中断和拷贝，降低了开销。用户态驱动采用轮询模式。轮询模式效率远高于中断模式，减少了中断模式上下文切换的开销。这部分操作需要在启动程序之前加载 uio 模块。

DPDK 使用 mempool 管理内存。Mbuf、ring、mempool 是 DPDK 内存管理的三个基本结构，有三个库与之相对应，分别是 Mbuf Library、Ring Library、Mempool Library。Mbuf 是数据的载体，一个 mbuf 存储一个报文信息，是报文

处理的基本单位。Ring 是一种支持多生产者多消费者的环形无锁队列结构，它采用比较和交换来避免使用锁，加速了访问。虽然 ring 无锁地支持多生产者和多消费者，但冲突访问会增加比较和交换的次数，所以依然应尽量避免冲突的发生。Mempool 是内存申请和管理的结构，能够让存储对象均匀的分布在所有内存通道中，并为每个核配备一个 cache，减少多核访问同一个 ring 时的冲突，加速报文对象的处理。三者构成了 DPDK 收发报文的基础。Mbuf、ring、mempool 之间的逻辑关系是 mempool 使用 ring 结构存储 mbuf。在图 2-1 b)中，为网卡接收队列和发送队列配置一个 ring 结构，可以批量接收和发送报文。一个报文存储在一个 mbuf 中，而一组 mbuf 存储在一个 ring 结构中。这部分对 DPDK 使用者表现为内存申请接口和收发报文接口，内存申请接口对应 mempool 的创建和配置，收发报文接口需要配置网卡队列和 CPU 的对应关系，而不需考虑 ring 等结构的细节。VPN 网关的上层功能模块和收发包接口的纽带是 mbuf。收包接口将 mbuf 交给上层功能模块处理，而上层功能模块将封装好的 mbuf 交给发接口发送。

为解决内核协议栈低效问题以及数据在内核态与用户态多次切换带来的开销，DPDK 平台上层逻辑全部在用户态实现。同时，DPDK 平台使用用户态驱动，绕开了内核协议栈，输出是链路层报文，并且不提供用户态协议栈，其上层逻辑也必须在用户态实现。

DPDK 平台上层是精简的用户态基础协议栈。用户态基础协议栈用于解析报文外层封装，是内核协议栈的替代。用户态基础协议栈上层是 VPN 功能逻辑。本文的 VPN 网关主要实现两种 VPN 功能，一种是连接转发功能，实现 VPN 客户端之间的通信；一种是代理转发功能，实现 VPN 客户端通过 VPN 网关访问外部网络。由于使用了 DPDK 平台，VPN 网关的功能也需要在用户态实现。

2.4 本章小结

本章主要研究如何将 DPDK 高性能报文处理平台结合用户态协议栈的框架应用到 VPN 网关中。VPN 使用隧道技术进行通信，根据隧道内层报文目的地址，可以将本文的 VPN 网关功能分为两种，一种称为连接转发功能，一种称为代理转发功能。VPN 性能评价指标包括 UDP 性能、TCP 性能包、时延和 jitter。DPDK 在网卡性能、驱动模式和内存管理等方面采用了多种加速技术，构成了高性能的基础，而 mempool 使用 ring 存储 mbuf 则构成了 DPDK 内存管理的基础。由于 DPDK 的输出是链路层报文而又不提供上层协议栈，故需要用户自行

建立用户态协议栈。VPN 网关的框架由 DPDK 高性能报文收发平台、用户态基础协议栈和 VPN 功能模块组成，收发报文绕过了内核协议栈，所有 VPN 功能都在用户态实现。本章介绍了 VPN 网关的框架，介绍了使用用户态协议栈的必要性和带来的性能提升，接下来两章主要介绍在用户态实现 VPN 两种功能时遇到的问题，一个是 VPN 路由查找问题，一个是用户态 NAT 问题。

第 3 章 VPN 路由查找算法的研究与实现

本章主要讨论连接转发功能中的 VPN 路由查找问题，并且只涉及 IPv4 地址的查找。首先，将介绍 VPN 路由查找的原理以及与普通路由查找的差异。接着，优化内核路由算法给出基于 Patricia 树的 VPN 路由算法。然后，在该算法的基础上，结合 VPN 路由查找的特点，提出基于集中插入的改进方法和基于划分阶段的改进方法。最后，通过实验分析算法的性能和适用场景。

3.1 VPN 路由查找原理

在第 2 章中介绍了 VPN 网关的两个主要功能，其中之一是连接转发功能，这个功能实现了 VPN 客户端之间通过 VPN 网关进行通信。在星形 VPN 网络中，VPN 网关处于中心位置，VPN 客户端分别与 VPN 网关建立隧道。为保证安全性，各个隧道使用不同密钥信息对负载进行加解密。具体来看，在客户端 A 和客户端 B 的通信中，A 先将两次封装的报文发送到 VPN 网关，VPN 网关脱去外层封装，使用 A 的密钥解密内层报文，然后将解密后的内层报文用 B 的密钥加密，再次封装后发送给 B。这会引起三个问题。第一个问题是，当 VPN 网关接收到一个报文时，如何找出使用哪组密钥信息对报文进行解密；第二个问题是，解密后的内层报文使用的是虚拟地址，在转发报文时，如何根据虚拟目的地址找到目的端使用的密钥信息，来对原始报文再次加密；第三个问题是，在为报文转发进行外层封装时，如何根据内层报文的虚拟目的地址找到外层封装需要的实际地址。

其中，第一个问题需要解决如何标记一条 VPN 连接，在第 5 章中介绍。剩下的两个问题是本章关注的问题。将路由信息和密钥信息合并在一起，可以将这两个问题概括为如何根据内层报文的虚拟地址找到对应的路由信息和密钥信息。这就是 VPN 路由查找。

普通路由查找指的是根据目的地址和掩码信息查找下一跳的地址信息。VPN 路由查找和普通路由查找不同。不同点在于，第一，VPN 路由查找使用的是虚拟地址；第二，VPN 路由查找不仅要找到下一跳路由信息，还需要找到接收端的密钥信息。第三，普通路由查找可以将网络地址作为路由查找的目标，因为普通网络中一个网络地址的设备通常集中在一起，可以由同一路由抵达，

但在 VPN 内网中，同一网络地址下的设备可能在地理位置上距离很远，所使用的实际地址完全不同，所以一般不能将网络地址信息做为路由查找的目标。

3.2 基于 Patricia 树的 VPN 路由算法的研究

路由查找算法有很多种，Linux 内核实现了两种路由查找算法，一种基于 hash，一种基于 LC-trie。而 4.4 BSD-Lite 内核采用了基于 Patricia（Practical algorithm to retrieve information coded in alphanumeric，Patricia）树的算法。本文首先研究基于 Patricia 树的 VPN 路由查找算法，然后在此基础上结合 VPN 路由查找的特点进行改进。

Patricia 树是一种前缀树。Patricia 树将普通前缀树中的一连串的“独生子女”压缩成一个结点，提高了空间利用率。4.4 BSD-Lite 使用 Patricia 树构造路由表，待查找的 IP 地址和树中的 IP 地址关键字都被看作比特序列，其示意如图 3-1 所示。树中结点分为叶子结点和内部结点。叶子结点含有一个主机地址或网络地址作为关键字，如果叶子结点仅含有主机地址，而没有包含掩码，那么该叶子结点对应一个主机地址，如果叶子结点含有掩码，则表示网络地址，叶子结点中包含待查找的路由信息。内部结点一般只包含指导搜索的索引信息，包括左右孩子结点的索引和比较位，有些内部结点含有掩码信息，用于回溯匹配。比较位是 Patricia 树结点中较为特殊的结构，它因压缩内部结点而产生，若设它的值为 *t*，可以这样理解 *t* 的含义：如果结点是内部结点，则在该结点的左孩子和右孩子的关键字比特序列中，前 *t* 个比特位都相同；如果结点是叶子结点，则代表孩子结点关键字的比特位数。

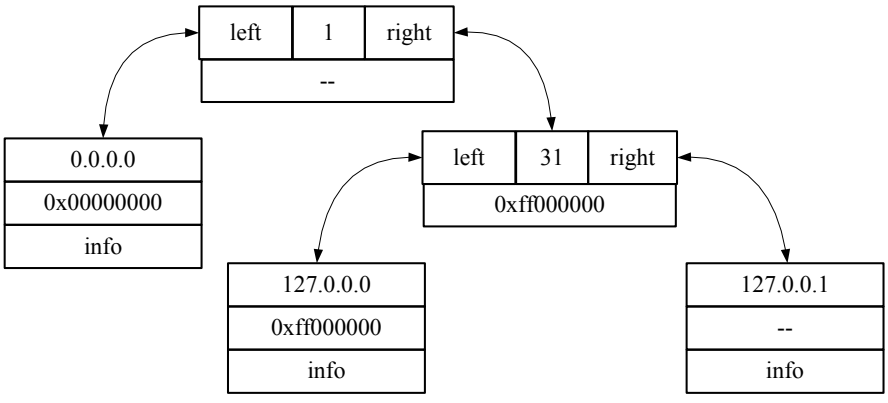


图 3-1 基于 Patricia 树的路由表示意图

VPN 路由查找较为特殊，不能采用网络地址作为查找目标，必须匹配虚拟

IP 的所有位。这样，Patricia 树中的结点都不需保留掩码信息，在查找过程中也不需要回溯，只要叶子结点没有匹配到，就可以判断为未查找到。

VPN 路由查找的关键字是虚拟 IP，将其看作比特序列，查找的内容是虚拟 IP 对应的客户端的路由信息和密钥信息，其中路由信息包括对应客户端的实际 IP、下一跳的物理地址等。由此，本文得到基于 Patricia 树的 VPN 路由插入、查找和删除算法。

3.2.1 基于 Patricia 树的 VPN 路由插入算法

基于 Patricia 树的 VPN 路由插入算法的主要流程示意如图 3-2 所示。

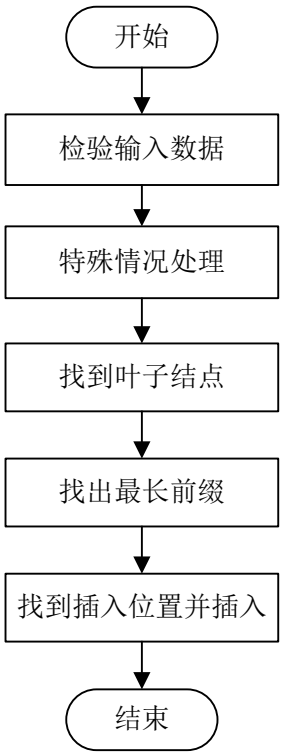


图 3-2 插入算法主要流程示意图

该插入算法可以描述为：

（1）检验输入数据。如果待插入的信息的索引为空，则判断为插入失败。

（2）特殊情况处理。如果 Patricia 树为空，表示树中还没有结点，则新建一个结点，将待插入信息插入到新建结点中，并将树的头索引指向新建结点。如果不为空，则转到（3）。

（3）找到叶子结点。将待插入信息关键字比特序列设为 k，将结点的比较位的值设为 t，从 Patricia 树的头结点开始，如果 $k[t]$（$k[0]$表示第 1 个比特位）

为 0，则向结点的左孩子继续搜索，如果 $k[t]$为 1，则向结点的右孩子继续搜索，这个判断的过程可以表示成图 3-3。一直搜索到叶子结点为止。如果叶子结点的关键字为 k，则需要使用待插入信息更新叶子结点中的信息，结束算法；如果叶子结点的关键字不为 k，则转到（4）。如果虚拟 IP 分配正确、回收正确，不会发生待插入信息的关键字和叶子结点中的关键字相同的情况，这里对叶子结点关键字值的判断是一种容错机制。

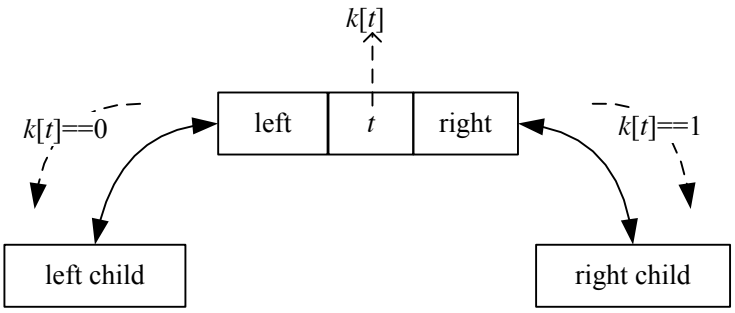


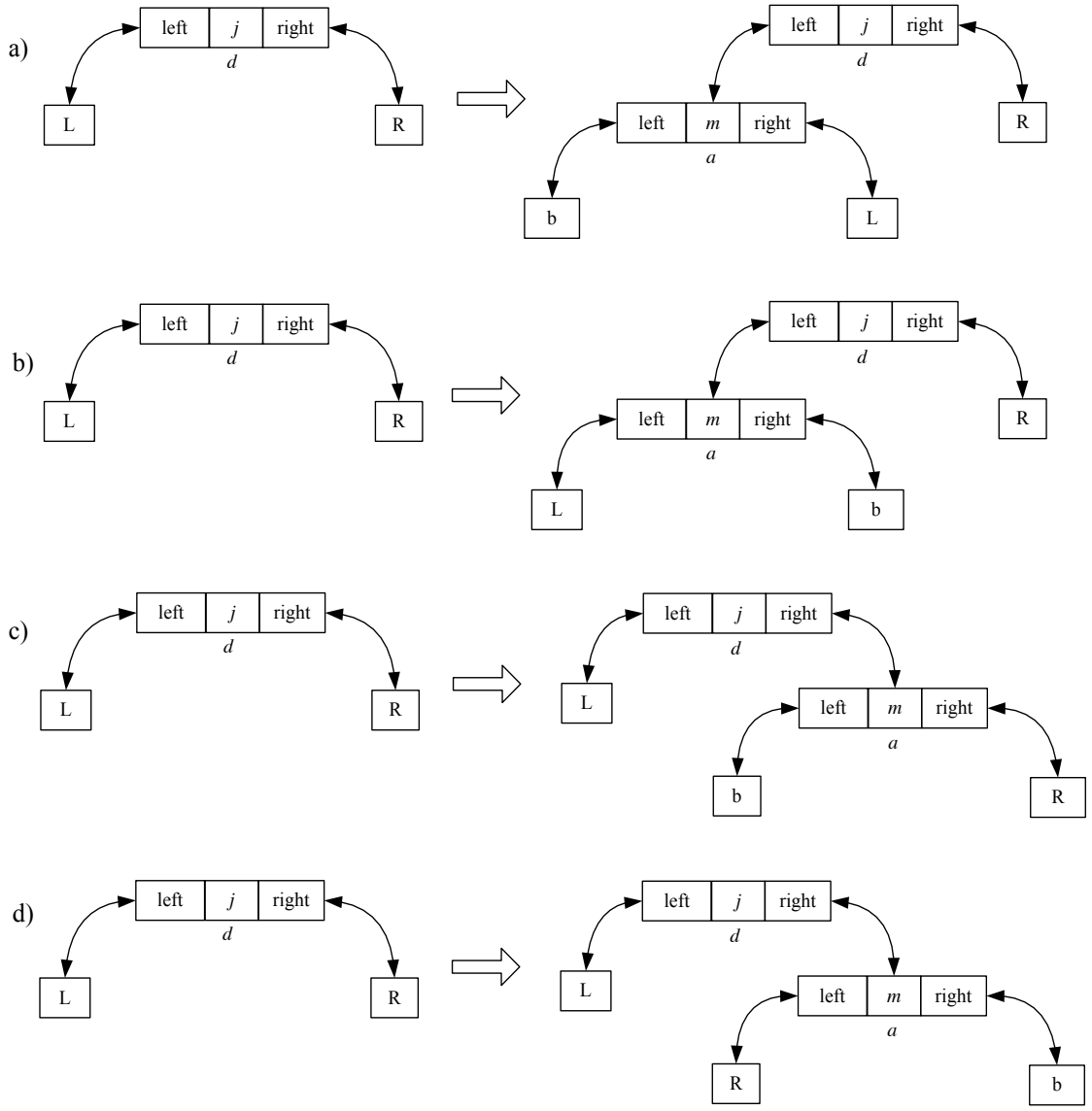
图 3-3 查找叶子结点分支判断示意图

（4）找出最长前缀。找出 k 和叶子结点关键字的最长相同前缀的位数，设为 m。可以通过按位比较实现，或通过先比较整字节再比较剩余位的方式实现。

（5）找到插入位置，插入待插入结点。从叶子结点的父结点开始，如果结点的比较位大于 m，则继续向上层搜索，直到搜索到头结点或搜索到比较位小于 m 的结点 d。新建一个内部结点 a，将这个结点的比较位设为 m，新建一个叶子结点 b，关键字比特序列为 k，将其存储信息设置为待插入信息。

如果搜索到的是头结点，则若 $k[m]$为 0，则将 a 的左孩子索引指向 b，将 a 的右孩子索引指向头结点，然后将头结点索引指向 a，若 $k[m]$为 1，则将 a 的右孩子指向 b，将 a 的左孩子索引指向头结点，然后将头结点索引指向 a。插入完成后，结束算法。

如果搜索到的是结点 d，设 d 的比较位为 j，分为四种情况，四种情况如图 3-4 所示。第一种情况，$k[j]$是 0，$k[m]$为 0，则 a 的左孩子索引指向 b，a 的右孩子索引指向 d 的左孩子，d 的左孩子索引指向 a。第二种情况，$k[j]$是 0，$k[m]$为 1，则 a 的右孩子索引指向 b，a 的左孩子索引指向 d 的左孩子，d 的左孩子索引指向 a。第三种情况，$k[j]$是 1，$k[m]$为 0，则 a 的左孩子索引指向 b，a 的右孩子索引指向 d 的右孩子，d 的右孩子索引指向 a。第四种情况，$k[j]$是 1，$k[m]$为 1，则 a 的右孩子索引指向 b，a 的左孩子索引指向 d 的右孩子，d 的右孩子索引指向 a。插入完成后，结束算法。



a) $k[j]$为 0 且 $k[m]$为 0　　b) $k[j]$为 0 且 $k[m]$为 1

c) $k[j]$为 1 且 $k[m]$为 0　　d) $k[j]$为 1 且 $k[m]$为 1

图 3-4 四种插入情况示意图

3.2.2 基于 Patricia 树的 VPN 路由查找算法

基于 Patricia 树的路由查找算法的伪代码如表 3-1 所示。由于树中不包含网络地址，查找时不需要回溯，搜索到叶子结点判断关键字是否相同即可。该查找算法的查找过程和插入算法中的查找叶子结点类似。根据待查找比特序列在比较位的值来判断向左子树查找或向右子树查找，一直查找到叶子结点，然后比较叶子结点的关键字和待查找关键字，如果相同则返回叶子结点索引，如果

不相同，则表示树中没有该关键字。在搜索过程中只比较了比特序列中的某些位，不能保证每一位都相同，但如果某个叶子结点的关键字是 k，那么按照这个步骤搜索到的叶子结点一定是关键字为 k 的结点。

表 3-1 VPN 路由查找算法伪代码

算法名称：基于 Patricia 树的 VPN 路由查找算法
算法输入：基于 Patricia 树的 VPN 路由表结构，记作 T 待查找虚拟 IP 比特序列 k
算法输出：若查找到，则输出含有关键字 k 的叶子结点的索引 若未找到，则输出 NULL

```
1. procedure vpn_route_lookup(T, k)
2.     if T is NULL then
3.         return NULL;
4.     end if
5.     y ← T->head;
6.     while y is not leaf then
7.         if k[y->cmp_bit_len] is 0 then
8.             y ← y->left_child;
9.         else
10.            y ← y->right_child;
11.        end if
12.    end while
13.    if y->key equal k then
14.        return y;
15.    end if
16.    return NULL;
17. end procedure
```

3.2.3 基于 Patricia 树的 VPN 路由删除算法

在删除 VPN 路由条目时，由于路由信息存储在叶子结点，故需要删除叶子结点，而叶节点代表了父节点在比较位的后一位的比特变化，删除叶子节点后，其父节点少了一种变化，不需要在该比特位分支，故父节点也需要删除。而父节点的删除会影响到父节点的父节点的孩子索引变化，故也需要考虑父节点的

父节点。这样，需要考虑三层结点的变化。

首先，是否存在三层结点有三种情况。第一种情况，只有一层结点，如图 3-5 所示，头结点指向待删除的叶子结点 *y*。此时，将头结点置空，并销毁 *y* 即可。第二种情况如图 3-6 所示，只有两层结点，头结点索引指向 *y* 的父结点。将待删除关键字比特序列记作 *k*，将 *y* 的父节点的比较位记作 *m*，此时，若 *k*[*m*]为 0，表示 y 是父节点的左孩子，则将头结点索引指向父节点的右孩子；若 *k*[*m*]为 1，表示 y 是父节点的右孩子，则将头结点指向父节点的左孩子；最后销毁 *y* 的父节点和 *y*，结束处理。第三种情况如图 3-7 所示，至少有三层结点。将 *y* 的父结点的父节点的比较位记作 *j*，此时，根据 *k*[*j*]、*k*[*m*]的值分为四种情形。其中，*k*[*j*]的值表示 y 的父节点的父节点的哪条孩子索引发生变化，*k*[*m*]的值表示 *y* 在父节点的哪条孩子索引上，需要将 *y* 的父节点的父节点发生变化的索引指向 *y* 的兄弟结点，然后销毁 *y* 的父节点和 *y*，四种情形的变化不再赘述。

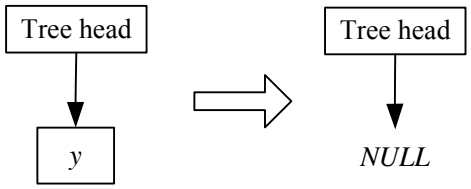
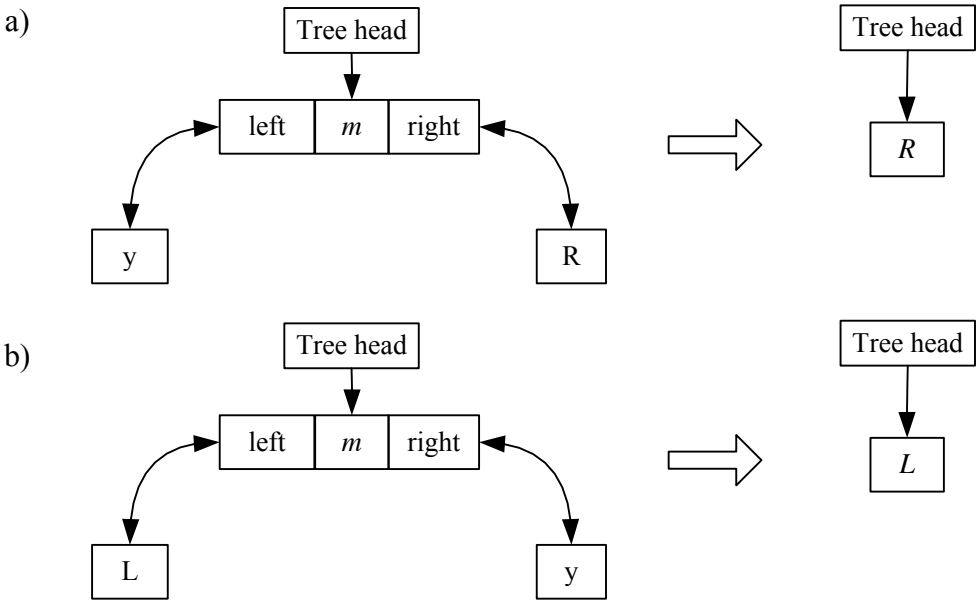
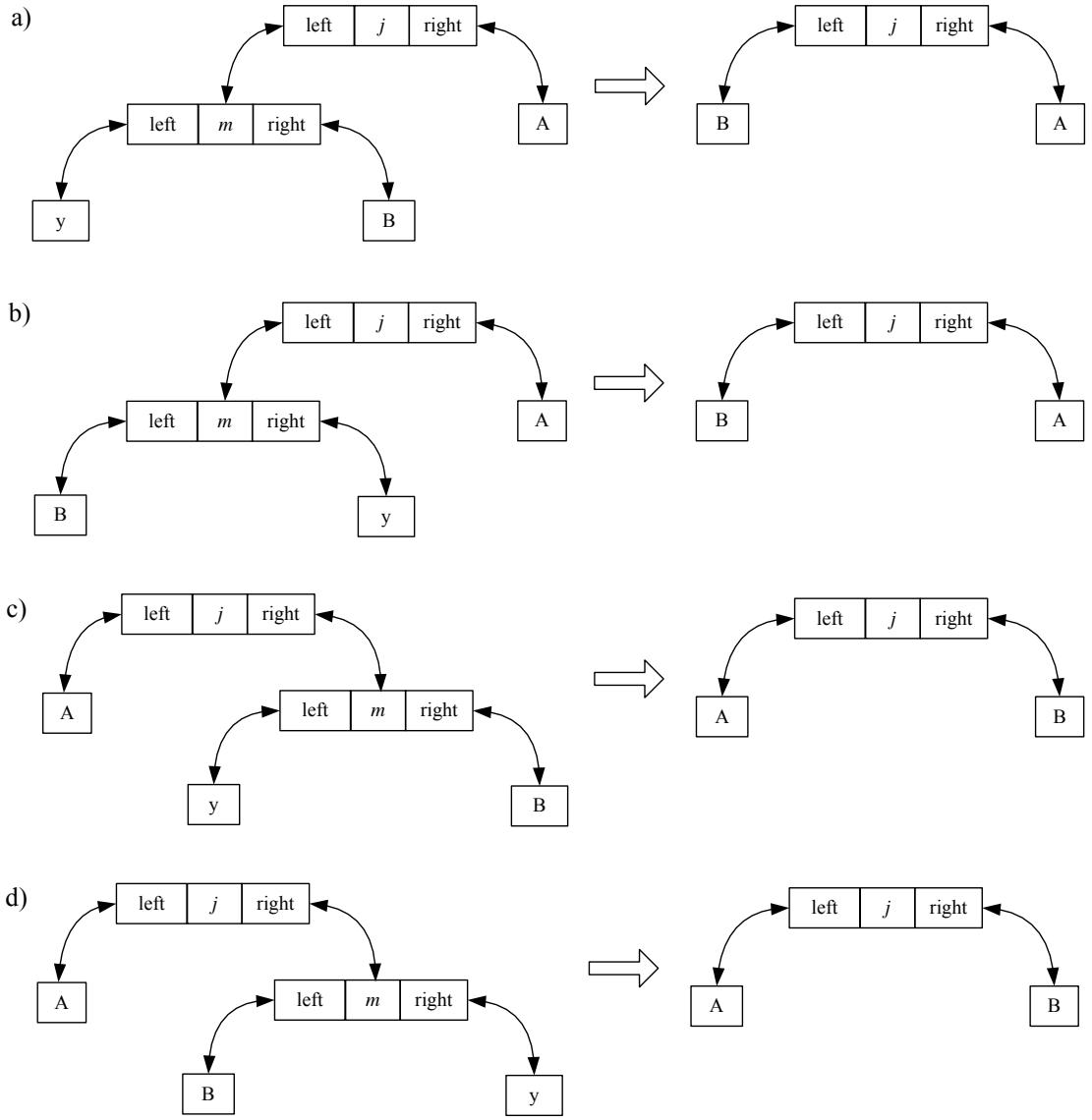


图 3-5 只有一层结点时删除示意图



a）*k*[*m*]为 0　　b）*k*[*m*]为 1

图 3-6 存在两层结点时删除示意图



a) $k[j]$为 0 且 $k[m]$为 0　　b) $k[j]$为 0 且 $k[m]$为 1

c) $k[j]$为 1 且 $k[m]$为 0　　d) $k[j]$为 1 且 $k[m]$为 1

图 3-7 至少存在三层结点时删除示意图

3.3 基于 Patricia 树的 VPN 路由算法的改进

基于 Patricia 树的 VPN 路由算法和普通的基于 Patricia 树的路由算法相比，利用了 VPN 路由查找不需要处理网络地址，减少了回溯查找，一定程度上降低了复杂度。结合 VPN 路由查找特点，在一定条件下，还可以进一步提升效率。

3.3.1 基于集中插入的改进方法

在插入、查找和删除三种算法中，插入和删除算法复杂度较高。基于 Patricia 树的 VPN 路由插入算法，经历了查找叶子结点、计算最长前缀、查找插入位置、分裂结点和插入信息等步骤；基于 Patricia 树的 VPN 删除算法，需要经过查找叶子结点和合并结点操作。插入对应客户端的接入操作，删除对应客户端的断开操作。在普通路由中，路由表是相对稳定的，但在多用户的 VPN 网络中，客户端的接入和断开较为频繁，插入和删除操作会影响 VPN 路由查找的效率。在普通的路由查找中，待查的关键字是目的 IP。普通路由查找的目的 IP 随机性强，无法预测，但 VPN 路由查找的目的 IP 是虚拟 IP，虚拟 IP 的范围有限，且规律性较强，一般从虚拟网段的第二个地址开始，从小到大分配。

基于虚拟 IP 的特点，可以采用“整段插入，不删除结点”的措施，事先定义好可能出现的网段，在程序初始化时，将虚拟 IP 先全部插入，在叶子结点中设置是否已使用的标志位，在需要插入新的路由条目时，查找到该叶子结点，将标志位设为已使用，并添加路由信息和密钥信息即可，而不必进行插入操作；在需要删除路由条目时，查找到该叶子结点，将标志位设为未使用即可，不必删除结点。如此，可以避免插入和删除操作中的结点分裂和插入删除操作，提高效率。“整段插入，不删除结点”的方式提升了插入和删除操作的效率，但一次性插入整个网段的虚拟 IP，会增加树的高度，如果该网段的在线客户端较多，这种方法很合适，但如果该网段的在线客户端较少，则会增加查找时间。而且这种方式下，如果希望动态增加网络地址，由于需要插入一个网段的所有虚拟地址，添加过程会非常耗时。

3.3.2 基于划分阶段的 VPN 路由算法的改进

上述“整段插入，不删除结点”的方式会增加树高，影响查找效率，可以进一步加以改进，使用基于划分阶段的方法，将 Patricia 树和 hash 表相结合。基于划分阶段的改进方法将虚拟 IP 的查找分为两个阶段，第一个阶段，查找虚拟网段，第二个阶段，定位网段中的具体虚拟 IP。在第一阶段的查找中，仍然使用 Patricia 树，在第二阶段的查找中，引入 hash 表。当虚拟网段固定时，虚拟 IP 的范围缩小了很多。一个 B 类的虚拟网段的虚拟 IP 个数最多是 65536 个，而一个 C 类虚拟网段的虚拟 IP 最多是 256 个。这样，可以使用一个直接定址的 hash 表来对应一个网段内的所有虚拟 IP，或部分虚拟 IP，可以将虚拟 IP 网

络字节序的最后一个或两个字节作为 hash 的关键字。再规定虚拟网段的掩码只能是 16 位或 24 位，并且 16 位的网络地址的有效部分不能是 24 位的网络地址的前缀，避免引起冲突。

这种方式下，外形上仍是一颗 Patricia 树，内部结点仍然只包含左右孩子索引和比较位。叶子结点对应一个网络地址，存有网络地址关键字、掩码和一个直接定址的 hash 表。Hash 表的每个位置代表该网段下的一个虚拟地址，存有路由信息、密钥信息和是否已使用标志位。使用链表将网络地址链接起来，这样，在动态插入网络地址时可以使用该链表判断是否会引起冲突。其结构和查找过程如图 3-8 所示。

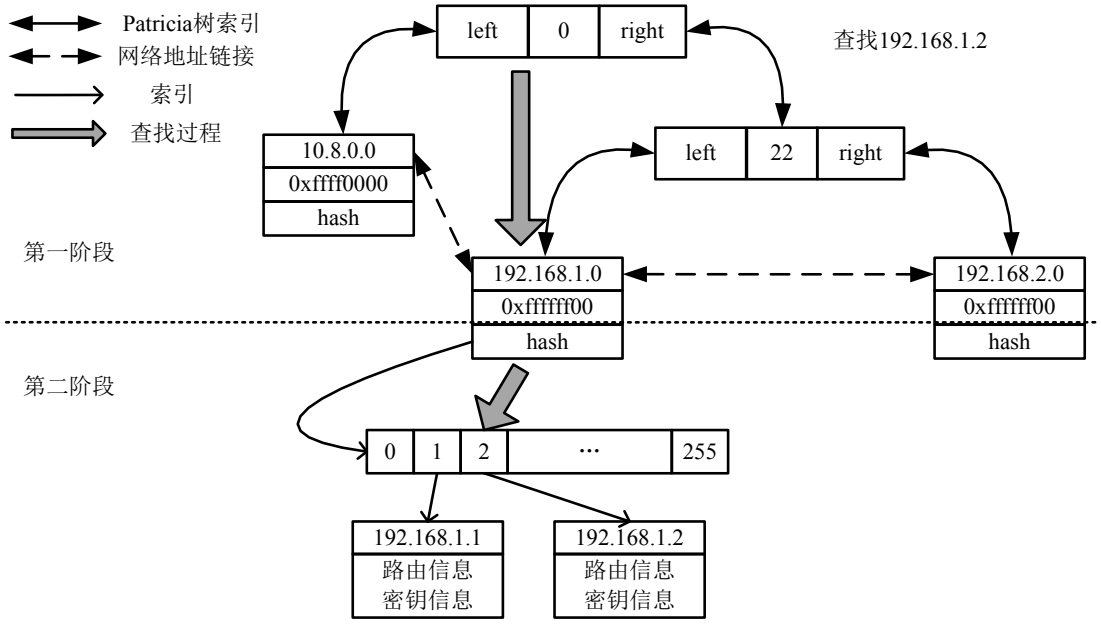


图 3-8 基于划分阶段的 VPN 路由查找示意图

插入新的网络地址的算法和之前的插入算法类似，但插入前需要验证其掩码是 16 位或 24 位，并遍历网络地址链表，验证待插入网络地址有效部分不能为已有网络地址的前缀，已有网络地址的有效部分也不能是待插入网络地址的前缀。插入叶子结点后，还要为其 hash 表进行初始化操作。

查找算法分为两个阶段。第一个阶段是查找网络地址，和基于 Patricia 树的查找算法类似，但查找到叶子结点后，需要和叶子结点的掩码做按位与运算，并判断是否和该网络地址相同。第二阶段是确定具体 IP 阶段，使用虚拟 IP 的网络字节序的最后一个或两字节做为 hash 的关键字，定位虚拟 IP 在 hash 表中的位置，如果该位置的标志位为已使用，则取出该位置的信息即可。插入路由

条目算法先按上述路查找算法查找到 hash 表中的位置，然后添加路由信息和密钥信息，并将标志位设为已使用。删除路由条目算法先使用查找算法查找到 hash 表中的位置，然后将标志位设为未使用。

3.4 VPN 路由查找算法测试和分析

本节通过测试比较基于 Patricia 树的 VPN 路由查找算法（下文简称 A 方法）、基于集中插入的改进方法（下文简称 B 方法）和基于划分阶段的改进方法（下文简称 C 方法）的性能和适用场景。

3.4.1 测试说明

测试的硬件参数如表 3-2 所示。

表 3-2 VPN 路由查找实验硬件参数

处理器	内存	操作系统
Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz	64G	Ubuntu 16.04.1 LTS

共进行 4 项测试，具体步骤如下。

测试 1：测试 Patricia 树路由表规模和查找时间的关系。具体如下：进行 5 组测试，各组测试都在 10.8.0.0/16 虚拟地址网段，从 10.8.0.1 开始依次插入，但插入规模不同，第 1 组到第 6 组分别插入 10 条、100 条、1000 条、10000 条、60000 条 VPN 路由信息。各组查找目标相同，都是 10.8.0.1 到 10.8.0.10 共 10 个关键字。每组测试查找 10000 次，记录总查找时间。

测试 2：测试一种极限情况，用户集中在同一网络地址。具体如下：对于 A、B、C 三种方法，在 10.8.0.0/16 虚拟地址网段，从 10.8.0.1 开始，依次插入 10000 条 VPN 路由信息，记录插入时间；从 10.8.0.1 开始，查找插入的 VPN 路由信息，记录总查找时间；从 10.8.0.1 开始，删除插入的 VPN 路由信息，记录总删除时间。

测试 3：用户集中在同一网络地址，在维持树高较为稳定的条件下插入删除 VPN 路由信息。具体如下：在 10.8.0.0/16 网段，从 10.8.0.1 开始插入 10000 条 VPN 路由信息。从 10.8.0.1 开始，依次遍历插入的 VPN 路由信息，删除该路由信息，然后再次插入该 VPN 路由信息。将上述遍历进行 10 次，记录 A、

B、C 三种方法的需要的时间。

测试 4：测试另一种极限情况，用户分散在多个网络地址，每个网络地址仅有两个用户。具体如下：网络地址分布从 192.168.0.0/24 到 192.168.255.0/24，每个网络地址下插入 192.168.X.1 和 192.168.X.2 两条 VPN 路由信息。遍历查找已插入的 VPN 路由信息，记录 A、B、C 三种方法的查找时间。

3.4.2 测试结果讨论和分析

图 3-9 是测试结果，其中图 a 到图 d 分别是测试 1 到测试 4 的结果。

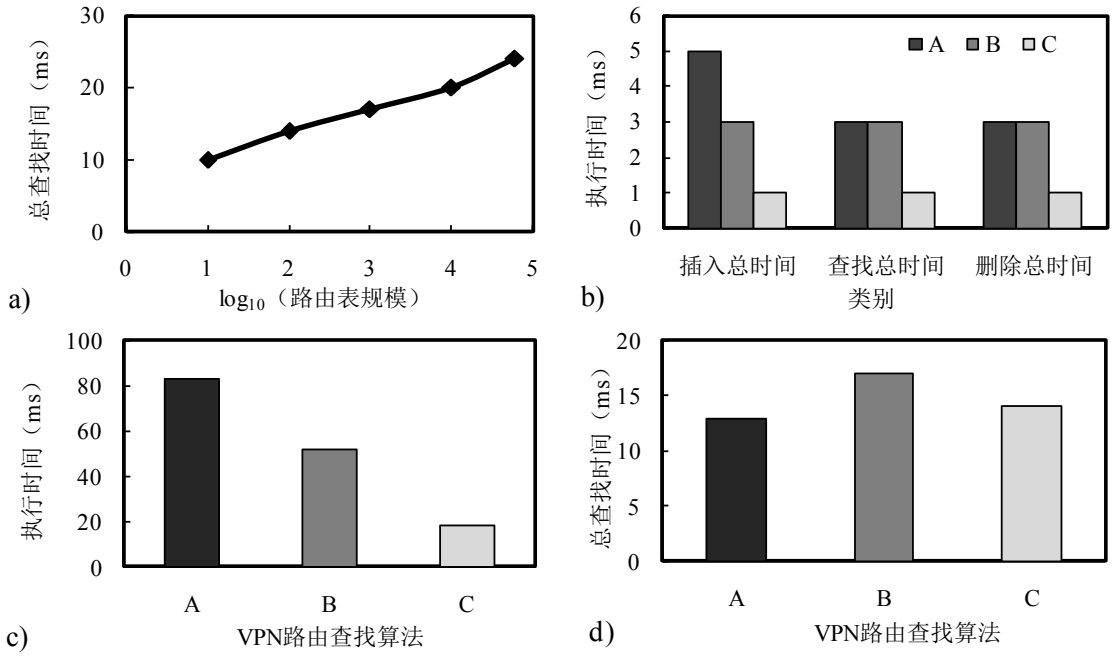
在图 3-9 a)中，由于路由表规模变化较大，故横坐标做了对数处理。基于 Patricia 树的路由表规模的对数和总查找时间呈近似的线性关系，路由表规模越大，查找时间越长。如果将对数换成实际值，将是一条起初陡峭上升而后迅速平缓的曲线。这是由 Patricia 树的性质引起的。Patricia 树可以看作一颗二叉树，每层能够容纳的结点是前一层的 2 倍，而查找时间和树高相关，叶子结点数目扩充一倍，只能增加一层树高，对树高的影响越来越低，对查找时间的影响也就越来越微弱。这表示，在用户较少的情况下，采用 B 方法增加了树高将会降低查找速度，这是 B 方法相对于 A 方法的劣势，但降低的幅度随规模越来越小，如果集中插入 10000 个虚拟 IP 而实际用户只有 10 人，那么其平均查找时间约是 A 方法的 2 倍。

在图 3-9 b)中，A 方法的插入时间明显高于查找时间，这是由于插入算法经历了多个步骤并且需要分裂结点造成的。A 方法的删除算法中包含了查找过程，但结果中删除时间和查找时间几乎相同，这是由于在查找过程中 Patricia 树高不变，但在删除过程中 Patricia 树高不断变低造成的。B 方法由于在初始化时已经将网络地址中的虚拟 IP 集中式插入，而且删除时并不直接删除结点，而是设置标志位，故其插入和删除过程主要是查找结点的过程，所以其插入和删除时间和查找时间接近。C 方法的插入、查找和删除过程相似性大，都分为查找网络地址阶段和确定具体 IP 地址阶段，所以插入、查找和删除的时间几乎相同。总体来看，在 10000 用户集中于单一网络地址的情况下，C 方法明显比 A 方法和 B 方法占优势，插入速度约是 A 方法的 5 倍，约是 B 方法的 3 倍，查找速度和删除速度约是 A 方法和 B 方法的 3 倍。

图 3-9 c)与图 3-9 b)的插入和删除时间意义不同。在图 3-9 b)中，插入从树为空开始到所有结点全部插入为止，删除从全部结点已插入开始，到全部结点都被删除为止。而在图 3-9 c)中，删除一个结点，接着又插入该结点，衡量的

是连接和断开较为均匀的条件下，插入和删除路由信息所需要的时间。从图 3-9 c)中可以看出，B 方法插入删除速度约是 A 方法的 1.6 倍，这是因为 B 方法采用集中插入，其插入时间接近查找时间，避免了分裂结点等操作，这是 B 方法相对于 A 方法优势的最大来源。但 C 方法更具优势，其插入和删除速度约是 A 方法的 5 倍，约是 B 方法的 2.9 倍。

图 3-9 d)模拟了少量用户分散于大量网络地址的情况。在这种情况下，B 方法由于插入了很多无用结点导致树的高度变高，查找时间比 A 方法长。C 方法相对于 A 方法的性能优势在于第二阶段的 hash 直接定址，而在用户分散于大量网络地址的情况下，C 方法的第一阶段查找网络地址在整个查找过程中所占的比例较大，第二阶段直接定址所占的比例较小，无法体现出查找优势，并且需要进行掩码操作，故性能略低于 A 方法。



a）基于 Patricia 树的路由表规模和总查找时间关系图

b）大量用户集中在单一网络地址条件下执行时间对比图

c）树高较稳定条件下插入删除总时间对比图

d）少量用户分散于多个网络地址条件下查找时间对比图

图 3-9 测试结果图

综上所述，在少量用户分散于大量网络地址的情况下，应选择基于 Patricia 树的路由查找算法，灵活性强，查找速度有优势。在大量用户集中于少量网络

地址并且插入和删除较为频繁的情况下，B 方法较 A 方法有一定优势，但不如 C 方法。在大量用户集中于少量网络地址的情况下，C 方法在插入、查找、删除方面都有明显优势。

3.5 本章小结

本章研究了在实现 VPN 网关连接转发功能时，需要根据虚拟 IP 查找路由信息和密钥信息的问题，即 VPN 路由查找问题。首先，优化了基于 Patricia 的路由算法，使其更加适用于 VPN 路由查找。然后根据 VPN 路由查找的特点提出了基于集中插入的改进方法和基于划分阶段的改进方法。最后，通过实验比较算法性能并分析了适用场景。实验表明，当少量用户分散于大量网络地址的情况下，基于 Patricia 树的 VPN 路由查找算法灵活性强，查找速度更快。而当大量用户集中于少量网络地址的情况下，基于划分阶段的改进方法更有优势。下一章将介绍在实现 VPN 网关代理转发功能时需要解决的用户态 NAT 问题。

第 4 章 用户态 NAT 的研究与实现

本章主要研究代理转发功能中的用户态网络地址转换（Network Address Translation，NAT）。首先，本文将介绍 NAT 的基本情况和分类，比较 VPN 网关的用户态 NAT 和普通 NAT 的异同；然后，说明 NAT 的映射本质并给出一种基于尝试的 NAT 核心算法；接着，将设计并实现一种 VPN 网关的用户态 NAT，并通过实验为其选择适当的 hash 函数；最后，通过实验验证该设计能够实现用户态 NAT 功能。

4.1 NAT 概述

4.1.1 NAT 简介

随着互联网的高速发展，用户迅速增多，IP 网络规模不断扩大，IP 地址面临枯竭问题。从长远角度看，解决该问题的方法是更换网络协议，扩大 IP 地址规模，采用 IPv6 地址协议。但 IPv4 网络已经在世界范围内实现，更换协议面临很多问题，是一个长期的过程，不是一蹴而就的。因而有很多短期方案应运而生。网络地址转换技术就是其中之一[42]。

NAT 将 IP 地址从一个地址域转换到另一个地址域，可以完成内部私有 IP 地址和公网 IP 地址之间的转换，实现内网主机和外网主机的通信。NAT 技术允许不同的私有地址映射到同一个公网地址，减少了公网地址的分配，实现了网络地址的复用，缓解了 IP 地址资源的短缺。同时，NAT 能够起到隐藏内网细节的作用，有一定的防护作用。

但 NAT 也存在一些缺点[43]。第一，NAT 和协议紧密相关，由于需要修改地址信息，如果协议的负载中出现 IP 或端口，那么 NAT 也需要处理负载。NAT 和协议的相关性造成了 NAT 实现的复杂性，很难保证 NAT 能够转换所有协议。第二，NAT 一般是外向型的，需要有内网主机先发出请求，才能收到外部主机的回复，外部主机通常不能直接向内网发出请求，这会导致一些应用失效。第三，NAT 提供者和 NAT 使用者一般呈星形结构，NAT 提供者的单点故障会导致整个网络不可用。第四，NAT 也带来一些安全问题，例如隐藏在 NAT 后的网络攻击不易追溯，不加控制的 NAT 实现容易受到攻击等。但现阶段，NAT 仍是一项必不可少的技术。

4.1.2 NAT 分类

NAT 需要修改报文的地址信息，其实现和协议相关，实现方式有很多种。在 NAT 相关的 RFC 文件中，并没有明确规定 NAT 的实现方式。但提及了 NAT 的分类。

在文献[44]中，将 NAT 分成了传统网络地址转换，双向网络地址转换，两次网络地址转换，多保障网络地址转换。其中，双向网络地址转换、两次网络地址转化、多保障网络地址转换是为解决传统 NAT 的问题而采取的特殊方法，本文不做讨论。传统网络地址转换又称外向型网络地址转换，分为如下两类。

（1）基本网络地址转换：只转换 IP，而不处理端口号。外部 IP 地址充足的情况下，为每个内部主机静态分配一个外部地址，或当内部主机有访问外部主机请求时动态从地址池中分配一个外部地址。

（2）端口地址转换：不仅转换 IP，还要转换端口号。这种技术允许多个主机公用一个 IP 地址。NAT 通常指这种技术。

在文献[45]中，将 UDP 协议的 NAT 实现分成了四类，尽管该分类针对 UDP 协议，但依然有很大的参考意义。

（1）全锥型：相同的内部 IP 地址和端口映射到相同的外部 IP 地址和端口，并且，任意外部主机可以通过该映射地址向该内部主机发送报文。

（2）受限锥型：相同的内部 IP 地址和端口映射到相同的外部 IP 地址和端口，但只有内部主机访问过的外部 IP 才能向内部主机发送报文。

（3）端口受限锥型：相同的内部 IP 地址和端口映射到相同的外部 IP 地址和端口，但只有内部主机访问过的外部 IP 和端口二元组才使用相同二元组向内部主机发送报文。

（4）对称型：同一个源地址、源端口、目的地址、目的端口四元组映射到同一个外部 IP 和端口，四元组中有任何一个元素不同，都将被映射到不同 IP 地址和端口。只有四元组访问的 IP 和端口二元组才能向该内部 IP 和端口发送报文，并且只能向该 IP 和端口发送报文。

在实现 NAT 时，选择合适的实现方法和实现策略很重要，需要考虑以下方面：第一，选择支持的协议，通常需要支持 TCP、UDP。第二，端口复用程度，用户量越少，复用的需求程度越小，实现方法就能够越简洁。第三，映射的层次，根据映射的主体不同，分为主机映射到端口，端口映射到端口，协议映射到端口和流映射到端口。第四，外部流入报文的准入规则。

4.1.3 VPN 网关用户态 NAT

当内网中的一台主机希望访问外网主机时，由于内网主机使用内网 IP 地址，故不能直接和外网通信，此时，就需要 NAT 来进行 IP 地址的转换。VPN 客户端使用虚拟 IP 进行通信，可以看做内网，当 VPN 的客户端希望通过 VPN 网关访问外网主机时，就需要 VPN 网关提供 NAT 支持。

传统 VPN 网关使用 iptables 来完成 NAT。iptables 的核心是内核中的 netfilter。但在本文基于 DPDK 的 VPN 网关中，报文接收后直接由用户态的 VPN 网关进程进行处理，绕开了内核协议栈，无法通过 iptables 完成 NAT。所以需要研究如何在用户态实现 NAT。用户态意味着不能使用内核已有的模块，需要自行实现。用户态 NAT 意味着需要自行构建 NAT 的映射关系，处理地址信息的转换，处理报文的解析。

VPN 网关中的用户态 NAT 和普通 NAT 有一些区别。一是，进行 NAT 转换的不是外层的报文头部，在 NAT 前需要先脱去报文的外层封装，将内层负载进行 NAT 转换；二是，普通 NAT 在转换由外网流向内网的报文时，需要确定内网地址，但 VPN 网关用户态 NAT 不仅要确定内网地址，还需要确定报文目的地址对应的客户端的 VPN 连接信息，如密钥等。但这些区别也可以看做是独立于 NAT 之外的处理，NAT 的核心过程并没有变，依然是映射关系的处理。

4.2 NAT 核心算法的研究

4.2.1 NAT 的本质

NAT 的核心是转换，合理的转换需要依靠映射。对于从内网向外网发出的报文，将其报文的源 IP，源端口，协议，目的 IP，目的端口五元组记作<*s_IP, s_port, out_pro, d_IP, d_port*>将该五元组集合记作 *OUT_5_TUPLE_SET* 或{<*s_IP, s_port, out_pro, d_IP, d_port*>}，将网关用于 NAT 的 IP 和端口二元组记作<*n_IP, n_Port*>，将该二元组集合记作 *SNAT_2_TUPLE_SET* 或{<*n_IP, n_Port*>}。网关需要根据报文信息确定 NAT 地址信息，这可以定义为由集合 *OUT_5_TUPLE_SET* 到集合 *SNAT_2_TUPLE_SET* 的映射（4-1）：

$$f : OUT\_5\_TUPLE\_SET \rightarrow SNAT\_2\_TUPLE\_SET \tag{4-1}$$

对于从外网流向内网的报文，将其报文的源 IP（外部 IP），源端口，协议，目的 IP，目的端口五元组记作<*o_IP, o_port, in_pro, gw_IP, gw_port*>，将该五

元组集合记作 *IN_5_TUPLE_SET* 或{<*o_IP, o_port, in_pro, gw_IP, gw_port*>}，将网关内网的内部 IP、端口二元组记作<*i_IP, i_port*>，将该二元组集合记作 *DNAT_2_TUPLE_SET* 或{<*i_IP, i_port*>}。网关需要根据报文信息，来确定内网地址信息，这可以描述为从集合 *IN_5_TUPLE_SET* 到集合 *DNAT_2_TUPLE_SET* 的映射（4-2）：

$$g: IN\_5\_TUPLE\_SET \rightarrow DNAT\_2\_TUPLE\_SET \tag{4-2}$$

NAT 实现的核心是找出映射规则，使得转换后的报文在发送到目的主机后，目的主机返回的报文能够找到正确的内部主机。也就是说，需要找到上述规则 *f* 和 *g*，使得：

$\forall < s\_IP, s\_port, pro, d\_IP, d\_port > \in OUT\_5\_TUPLE\_SET$，都有式（4-3）成立。

$$\begin{aligned} &g(< d\_IP, d\_port, pro, f(< s\_IP, s\_port, pro, d\_IP, d\_port >) >) \\ &=< s\_IP, s\_port > \end{aligned} \tag{4-3}$$

在前面的小节中提到了 NAT 的分类，包括基本网络地址转换，端口地址转换，三种锥型网络地址转换还有对称网络地址转换。从式（4-3）来看，这些分类实际是在寻找规则 *f* 和 *g* 的过程中，考虑的参数个数不同，例如，基本网络地址转换只需考虑的 *s_IP*，其他参数的变化不予考虑，而对称 NAT 则需考虑所有参数。

4.2.2 NAT 核心算法的设计

将映射（4-1）写成映射（4-4）的形式：

$$\begin{aligned} &f: \{< s\_IP, s\_port, out\_pro, d\_IP, d\_port >\} \\ &\rightarrow \{< n\_IP, n\_Port >\} \end{aligned} \tag{4-4}$$

将映射（4-2）写成映射（4-5）的形式：

$$\begin{aligned} &g: \{< o\_IP, o\_port, in\_pro, gw\_IP, gw\_port >\} \\ &\rightarrow \{< i\_IP, i\_port >\} \end{aligned} \tag{4-5}$$

结合实际的报文情况，可以发现，能够控制的变量只有 *gw_IP* 和 *gw_port*，他们分别是网关用于 NAT 转换的 IP 和端口，其他变量都是不能修改的。所以，关键点是找一个合适的 *gw_IP*，*gw_port* 二元组使得式（4-3）成立。

式（4-3）提供了一个核心思路：在考虑规则 *f* 时，必须考虑规则 *g*，也就是说，在考虑报文流出的映射规则时，必须保证报文流入规则能够使报文的响

应返回时找到正确的内部地址。

也就是说，当在为五元组<*s_IP, s_port, out_pro, d_IP, d_port*>寻找找合适的 *gw_IP*，*gw_port* 二元组时，要考虑报文流入规则 g。由此，得到一个尝试型的 NAT 地址分配算法。主要思路是：当需要分配一个 *gw_IP*，*gw_port* 二元组时，先判断如果使用该二元组，是否能够使响应报文找到正确内部主机，即是否能够保证 *g* 的每个原象都有唯一确定的象与之相对应，也即对于当前的 *g*，有：

$$(< d\_IP,\ d\_port,\ out\_pro,\ gw\_IP,\ gw\_port >,< gw\_IP,\ gw\_port >) \notin g$$

如果能够保证，则分配该二元组，如果不能保证，则尝试其他二元组。上述算法的伪代码如表 4-1 所示。

表 4-1 NAT 地址分配算法伪代码

算法名称：NAT 地址分配算法	
算法输入：流出报文的五元组〈*a, b, c, d, e*〉，记作 *t* 网关用于 NAT 转发的 IP 和端口二元组<*h, i*>的集合，记作 *S* 前述映射规则 *g*	
算法输出：适合于 *t* 的 NAT 转地址二元组 s_j	
1. procedure find_nat_addr(*t, S, g*)	
2.	**if** either *t* or *S* is NULL **then**
3.	return NULL;
4.	**end if**
5.	**for** each member value s_j of *S* **do**
6.	**if** $(< d,\ e, c,\ s_j >,< s_j >) \notin g$ **then**
7.	return s_j;
8.	**end if**
9.	**end for**
10.	return NULL;
11.end procedure	

上述算法是一次 NAT 地址分配的过程。在 NAT 地址分配的基础上，可以得出向外部流出报文的处理算法，该算法的伪代码如表 4-2 所示。具体过程描述如下。

（1）输入参数检验。如果流出报文五元组<*a, b, c, d, e*>为空，或网关可用的 NAT 地址二元组集合为空，则结束处理。否则，进入（2）。

（2）如果映射 *f* 中有流出报文五元组的象，则使用该象做为 NAT 地址，

并对流出报文完成网络地址转换处理，结束处理。否则进入（3）。

（3）使用表4-1所示的NAT地址分配算法分配一个合适的NAT地址<h, i>。如果分配地址为空，则结束处理。否则，进入（4）。

（4）更新 NAT 规则。在映射 *f* 中添加(<*a*, *b*, *c*, *d*, *e*>, <*h*, *i*>)，在映射 *g* 中添加(<*d*, *e*, *c*, *h*, *i*>, <*a*, *b*>)。进入（5）。

（5）使用<*h*, *i*>作为 NAT 地址，对流出报文完成 NAT 网络地址转换处理，结束处理。

表 4-2 流出报文处理算法伪代码

算法名称：	流出报文处理算法
算法输入：	流出报文的五元组〈*a*, *b*, *c*, *d*, *e*〉，记作 *t* 网关用于 NAT 转发的 IP 和端口二元组<*h*, *i*>的集合，记作 *S* 前述映射规则 *f* 前述映射规则 *g*
算法输出：	无

```
1. procedure out_packet_process(t, S, f, g)
2.     if either t or S is NULL then
3.         return;
4.     end if
5.     if f(<a,b,c,d,e>) is not NULL then
6.         use f(<a,b,c,d,e>) as nat address and do packet nat process;
7.         return;
8.     end if
9.     <h, i> = find_nat_addr(t, S, g);
10.    if <h, i> is NULL then
11.        return;
12.    end if
13.    add (<a, b, c, d, e>, <h, i>) to f;
14.    add (<d, e, c, h, i>, <a, b>) to g;
15.    use <h, i> as nat address and do packet nat process;
16.    return;
17.end procedure
```

在该算法中，NAT 的映射规则 *f* 和 *g* 不是静态的，而是动态变化的。该算

法是 NAT 规则增加的过程，还需要考虑 NAT 规则的删除。NAT 规则的删除没有很好的方法，大部分采取的是超时删除的方式，本文也采用这种方式。该算法使用了映射来表述，抽象程度较高，适应性强，但同时，距离实际还有一定距离，在实现算法时还需要考虑很多细节，例如当一个连接终断时，如何删除所有相关规则等，两个规则各自还需要考虑哪些参数等。这些内容在下文讨论。

4.3 VPN 网关用户态 NAT 的设计

4.3.1 用户态 NAT 总体结构设计

图 4-1 给出了用户态 NAT 的总体架构。各模块的功能设计如下：

（1）NAT 规则：用户态 NAT 的数据结构，用于存储 NAT 的规则信息，实现前面小节中的映射规则 f 和 g。

（2）流出报文处理：接收报文并判断是否是流出报文。判断为内部向外部流出的报文时，对 NAT 规则模块进行插入、更新、查询操作，完成源地址转换功能，为报文流入提供处理规则，然后转发 NAT 后的报文。

（3）流入报文处理：外部报文向内部部流入时，这个模块需要对 NAT 规则模块进行查询操作来完成外部报文的过滤，并将准许进入内部的报文按照 NAT 规则进行目的地址转换。同时，准许进入的报文将要发送到 VPN 客户端，所以还需要根据目的地址完成 VPN 连接信息的查找以及转发。

（4）NAT 规则删除：当规则不再使用时，删除 NAT 规则。使用超时判断规则是否在使用中。

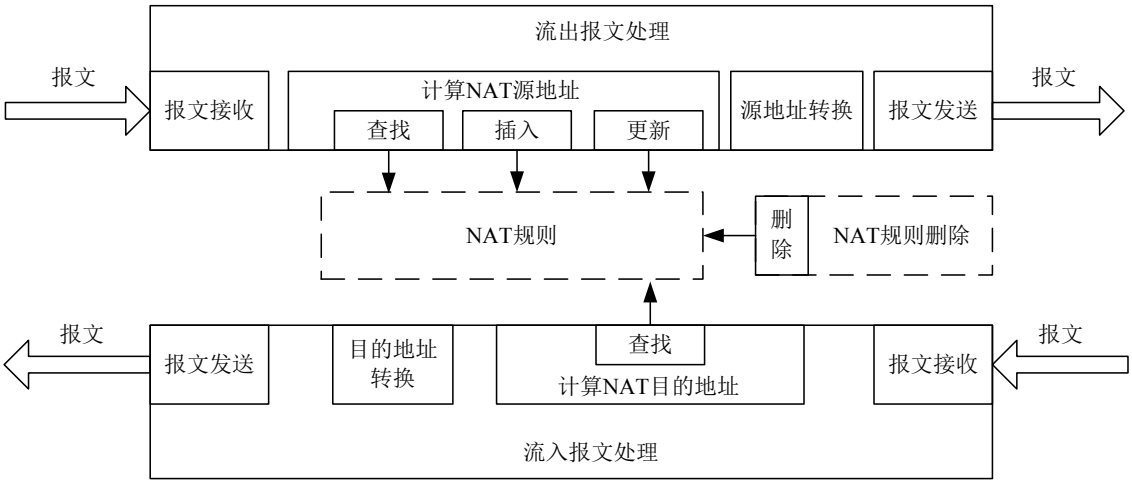


图 4-1 用户态 NAT 数据结构示意图

4.3.2 用户态 NAT 数据结构设计

用户态 NAT 数据结构设计主要分为两部分内容，第一部分结合 NAT 分类小节提出的设计 NAT 时需注意的方面，对用户态 NAT 进行分析，第二部分基于分析给出用户态 NAT 数据结构的设计。

（1）用户态 NAT 分析

NAT 有多种设计，RFC 文件并没有明确 NAT 实现方法。本文在设计用户态 NAT 时，主要从以下四个方面进行分析：

1）协议支持方面，先实现最基本的 TCP 和 UDP 协议。

2）端口复用方面，考虑端口多重复用。在最初的设计当中，考虑到，如果用户的联网端口平均 20 个以内，网关一个外部 IP 有 6 万多端口可以分配，也就能为约 3 千主机提供服务。这样，就可以采用全锥型网络地址转换的设计，即为内部主机的端口排他性地分配 NAT 地址，映射规则用数组就可以实现，设计和处理简洁明了。但在实现后发现，一个主机的联网端口不是几十的规模，即使为其分配 100 个端口，依然很快就会耗尽，毕竟一个主机的端口总量是 65535 个。所以，一般情况下，端口多重复用是一个必须考虑的问题。

3）准入规则方面，只允许内部主机正在访问的外部主机向内部发送报文。

4）映射层次方面，实际就是如何标记一个规则的主体。对于规则 *f*，Linux 内核采用流作为规则的主体，即流映射到端口。采用流映射到端口，需要五个关键字。但考虑到规则关键字越长，匹配耗时越长，故在选择规则主体时尽量不采用流。而在一般情况下，内部主机作为请求的发起方，一个端口的同一个协议在同一时间只能访问同一个目的地址，故对于规则 *f*，关注 *s_IP*，*s_port*，*out_pro* 三个参数，最好采用<*s_IP, s_port, out_pro*>映射到<*gw_IP*，*gw_port*>，即协议映射到端口，这样，当目的地址发生变化时，并不影响 *f* 中的规则，只需更新 *g* 中规则即可。而协议类别又十分有限，故先采取端口到端口的映射，然后再区分协议。对于规则 *g*，出于准入规则的考虑，*o_IP*，*o_port*，*in_pro*，*gw_IP*，*gw_port* 五个参数都需要关注，采用<*o_IP, o_port, in_pro, gw_IP, gw_port*>映射到<*i_IP, i_port*>, 即流映射到端口。

（2）用户态 NAT 数据结构设计

映射这个概念容易使用 hash 表实现，所以，使用两个 hash 表来实现规则 *f* 和 *g*。规则 *f* 对应的 hash 表称作 h_f ，规则 *g* 对应的 hash 表称作 h_g 。结合用户态 NAT 的分析， h_f 的元素素结构如表 4-3 所示， h_g 的元素结构如表 4-4 所示。

表 4-3 h_f 元素结构表

名称	中文描述	字母代号
关键字	流出报文源 IP	s_IP
	流出报文源端口	s_port
非关键字	UDP 协议索引	udp_index
	UDP 协议索引时间戳	udp_timestamp
	TCP 协议索引	tcp_index
	TCP 协议索引时间戳	tcp_timestamp

表 4-4 h_g 元素结构表

名称	中文描述	字母代号
关键字	流入报文源 IP	o_IP
	流入报文源端口	o_port
	协议	in_pro
	流入报文目的 IP	gw_IP
	流入报文目的端口	gw_port
非关键字	实际目的 IP	i_IP
	实际目的端口	i_port
	VPN 连接索引	vpn_session

Hash 表的关键字就是规则定义域五元组中需要考虑的参数。而 hash 表中的存储内容除关键字和映射对象外，对于 h_f，还需存储协议索引；对于 h_g，流向内部的报文不仅需要确定内部的 IP，还需要确定 VPN 连接的参数，所以还需要存储 VPN 连接的信息的索引 *vpn_session*。

超时删除会引起三个问题。第一个问题是如何保证两个 hash 表的同步，因为两个 hash 表一个处理流出报文的映射，一个处理流出报文的映射，两者间是有关联的。这就要保证当 h_f 要删除一个<*s_IP, s_port, out_pro*>确定的条目时，h_g 能够同时删除相应条目。第二个问题是既然两个表是有关联的，那么应该由哪一方作为删除操作的主动方，哪一方作为删除操作的从动方。第三个问题是如何判断超时。针对第一个问题，为保证超时删除的同步，将 h_f 中存储的协议索引指向 h_g 中对应条目，用索引来保证同步。针对第二个问题，作为一次请求和响应的发起方和网关的保护方，内部主机应该作为主动方，也就是说，h_f 应该作为超时删除的主动方。针对第三个问题，作为超时删除的主动方，h_f 不仅是

一个 hash 表，还应该是一个有超时机制的 hash 表。h_f 应为每个<*s_IP, s_port, out_pro*>确定的条目设置一个时间戳，并将每个条目按时间从远到近链接起来，供超时判断使用。经过上述分析，可以将用户态 NAT 的数据结构表示成图 4-2。

Hash 表中一个重要部分是 hash 函数。Hash 函数会影响 hash 表的计算速度和散列程度。Hash 函数的比较将在下文介绍。

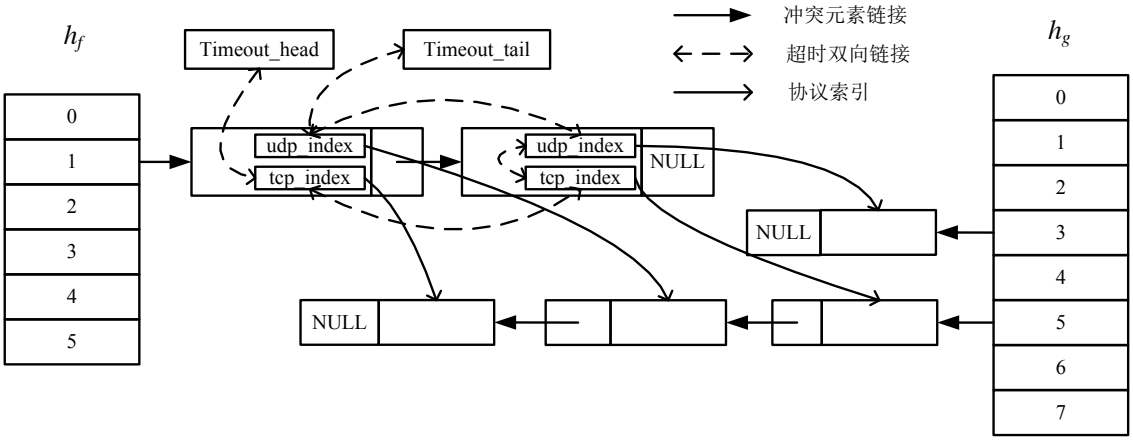


图 4-2 用户态 NAT 数据结构示意图

4.3.3 流出报文的处理

流出报文处理流程示意如图 4-3 所示。

将 NAT 地址分配算法中的规则 g 替换成 hash 表 h_g，并利用 hash 表性质，容易得到能够实际应用的 NAT 地址分配算法。其中遍历网关 NAT 地址的部分，有很多种方法，这里可以给出一种设计：对于一个可用于 NAT 的 IP，从 5001 端口开始尝试，直到 65535 端口，如果依然没有找到合适的二元组，则尝试下一个 IP。更新协议索引和删除超时元素在删除规则小节进行说明。

流出报文处理描述如下：

（1）解析出流出报文五元组

接收报文，判断所属的 VPN 连接，其索引记作 *vpn_session*，使用 *vpn_session* 对应的元素中保存的密钥对负载进行解密，解析解密后负载中报文的五元组，记作<*a, b, c, d, e*>，如果目的地址<*d, e*>不能在 VPN 路由表中找到，则认为是外部地址，判断为流出报文。

（2）计算 NAT 地址，修改 NAT 规则

这个过程可以概括为：如果已有对应规则，就从已有规则中取出 NAT 地址；如果没有对应规则，就建立对应规则，计算出 NAT 地址。

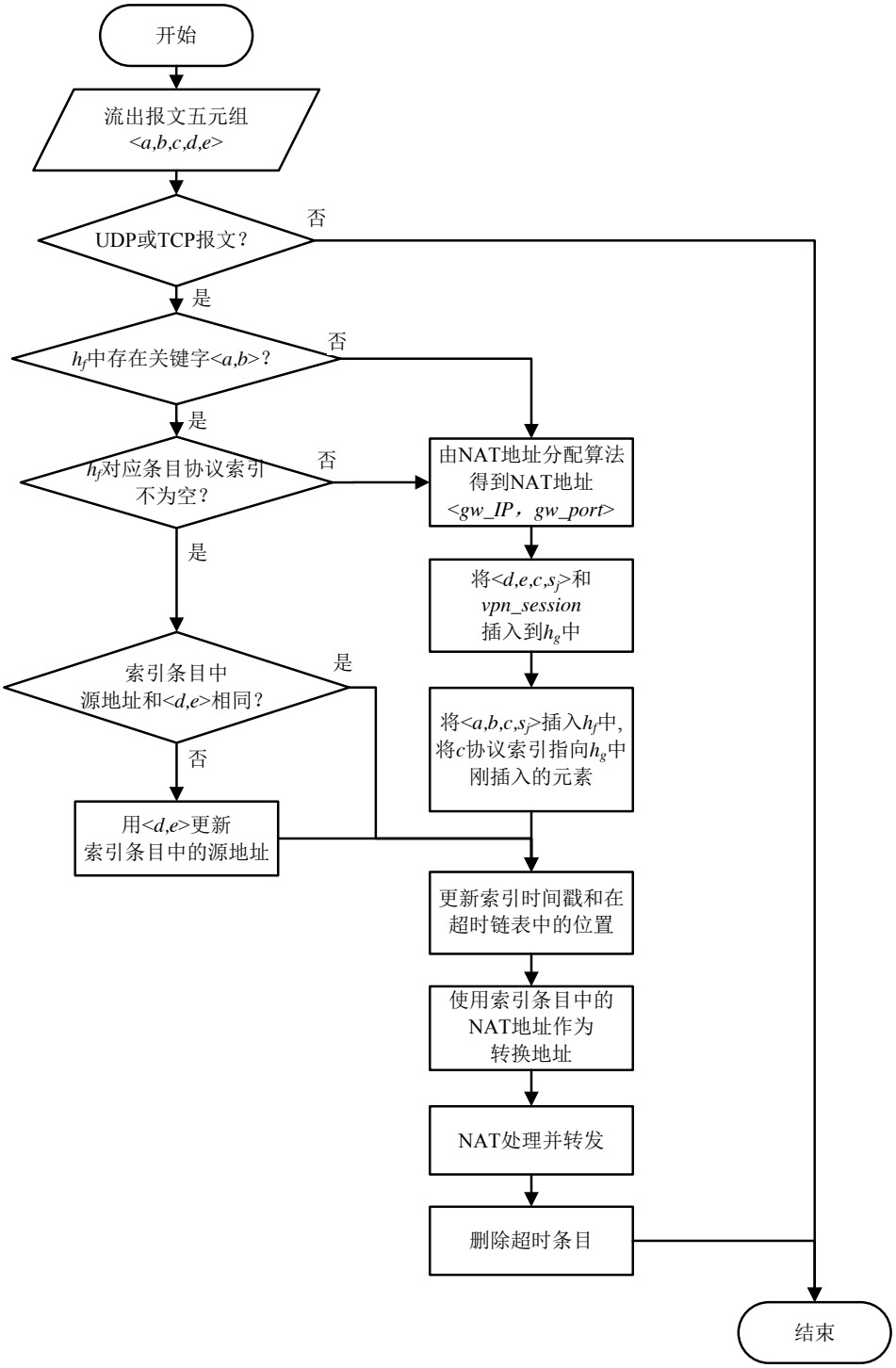


图 4-3 流出报文处理流程示意图

详细过程描述如下。对于流出报文，先判断<*a, b, c, d, e*>是否已经有对应的转换规则。判断已有对应转换规则需要经过顺序的四步判断：1）*c* 协议是 UDP 或 TCP 协议。2）h_f 中存在以<*a, b*>为关键字的元素 *m*。3）*m* 中，*c* 协议

索引不为空。4）该 *c* 协议索引对应的 h_g 中的条目的源地址是<*d, e*>。

若四个条件全部满足，则从 h_f 对应条目中取出 NAT 地址<g*w_IP, gw_port*>。

若 1），2），3）满足，4）不满足，进行更新操作。更新 h_g 中 *c* 协议索引指向的元素，将其源地址更新为<*d, e*>。

若 1），2）满足，3）不满足，则先使用 NAT 地址分配算法计算出合适的 NAT 地址<*gw_IP, gw_port*>。然后在 h_g 中插入以<*d, e, c, gw_IP, gw_port*>为关键字的元素<*d, e, c, gw_IP, gw_port, a, b, vpn_session*>，记作 *n*。再将 *m* 中的 NAT 地址更新为<*gw_IP, gw_port*>。最后将 *c* 协议索引指向 *n*，并更新 *c* 协议索引的时间戳。

若 1）满足，2）不满足，则先使用 NAT 地址分配算法计算出合适的 NAT 地址<*gw_IP, gw_port*>。然后在 h_f 中插入以<*a, b*>为关键字的元素 *t*，将 *t* 的 NAT 地址置为<*gw_IP, gw_port*>。再在 h_g 中插入以<*d, e, c, gw_IP, gw_port*>为关键字的元素<*d, e, c, gw_IP, gw_port, a, b, vpn_session*>，记作 *n*。最后将 *m* 中 *c* 协议索引指向 *n* 并更新 *c* 协议索引的时间戳。

若 1）不满足，由于目前只支持 UDP 和 TCP 协议，整个流出报文处理处理结束。

（3）源地址转换

使用上一步计算出的 NAT 地址修改传输层协议头的源端口，修改 IP 协议头源地址；重新计算传输层校验和并更新传输层协议头部校验和字段；重新计算 IP 协议校验和并更新 IP 协议头部的校验和字段。

（4）发送修改后的报文

先使用链路层协议进行封装，再将报文使用 DPDK 报文收发模块发送报文。在使用链路层协议封装时，需要填充下一跳 MAC 地址，此处采用静态的方法，统一填充为 VPN 网关服务器的网关的 MAC 地址。

4.3.4 流入报文的处理

流入报文的处理过程描述如下：

（1）解析出流入报文五元组：接收报文，解析出流入报文的源 IP、源端口、协议、目的 IP、目的端口五元组，记作<*a, b, c, d, e*>。

（2）计算实际目的地址：如果 h_g 中有以<*a, b, c, d, e*>为关键字的元素 *t*，则允许报文流入，其实际目的地址为 t 中记录的实际地址，记作<*m, n*>。如果 h_g 中没有以<*a, b, c, d, e*>为关键字的元素，则不允许报文流入，丢弃该报文。此

处如果不加限制，任何外部主机都可以通过<*d*, *e*>向 NAT 规则对应的内部某台主机发送报文，存在很大安全隐患。这里，匹配时对源地址做了限制，只有内部主机正在通信的地址、端口二元组才能向内部发送报文，降低了安全隐患。

（3）目的地址转换：使用上一步计算出的实际地址修改传输层协议头的目的端口，修改 IP 协议头目的地址；重新计算传输层校验和并更新传输层协议头部校验和字段；重新计算 IP 协议校验和并更新 IP 协议头部的校验和字段。

（4）加密报文：使用 *t* 中的 *vpn_session* 取出 VPN 连接使用的加密参数，加密上一步修改后的报文。

（5）发送报文：使用 *t* 中的 *vpn_session* 取出 VPN 连接使用的实际地址信息，将加密后的报文作为负载，添加 UDP 协议封装，添加 IP 协议封装，添加链路层协议封装。使用 DPDK 报文收发模块发送封装后的报文。

4.3.5 NAT 规则的删除

删除 NAT 规则采用超时机制。

（1）时间戳的更新：h_f 是超时删除的主动方，h_g 是从动方。故在流入报文处理时，在报文发送前需要更新报文在 h_f 相应规则中对应协议索引的时间戳，并将该索引在超时链表中的位置调整到超时链表的末尾。

（2）删除规则：大规模删除超时链表中的元素会影响报文处理，故将在每个 NAT 报文处理后删除超时元素。这样，需要删除的元素始终都较少，不会影响报文处理。删除方法具体描述为：从超时链表头部开始，将协议索引的时间戳和当前时间比较，判断协议索引是否超时，如果超时，删除协议索引对应的 h_g 中的元素，协议索引置空，删除超时链表中该元素；如果未超时，则删除结束。

4.3.6 Hash 函数的选择

Hash 函数是 hash 表的精髓。它根据 hash 元素的关键字计算出 hash 元素的存储位置。Hash 表元素的增、删、改、查都需要调用 hash 函数。而在用户态 NAT 的设计中，两个 hash 表是关键结构，被频繁操作。每个需要 NAT 的报文都至少需要操作一次 hash 表，也就是说，至少调用一次 hash 函数。因此，hash 函数的计算速度和散列程度对用户态 NAT 的性能有较大影响。

本小节通过实验评估 hash 函数的计算速度和散列程度，来为用户态 NAT 选择合适的 hash 函数。实验的使用的 hash 函数如表 4-5 所示。实验硬件参数如表 4-6 所示。

表 4-5 Hash 函数说明表

哈希函数	提出者	参考文献
BPHashs	--	Ref.[46]
DJBHash	Daniel Bernstein	Ref.[46]
CRC32	--	Ref.[47]
BKDRHash	B.Kernighan	Ref.[46]
BOB	Robert Jenkins	Ref.[48]
PJWHash	Peter J. Weinberger	Ref.[46]

表 4-6 Hash 函数实验硬件参数

处理器	内存	操作系统
Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz	65.7G	CentOS Linux release 7.2.1511

（1）实验数据

采用生成的模拟报文五元组进行实验。具体生成方法如下：将需要进行 NAT 的报文的源 IP、源端口、协议、目的 IP、目的端口五元组，记作<a, b, c, d, e>。a 的生成方法：虚拟地址从 10.8.0.1 到 10.8.39.55，共计 1 万个。b 的生成方法：1 到 65535 之间的随机值。为每个虚拟 IP 生成 1000 个随机端口。c 的生成方法：在 0 和 1 之间随机，随机到 0，将 c 设为 6（TCP 协议编号），随机到 1，将 c 设为 17（UDP 协议编号）。d 的生成方法：将 d 看成 4 个字节，每个字节的值为 0 到 255 之间的随机整数。e 的生成方法：在 1 到 1024 之间的随机值。

（2）实验步骤

将生成的五元组依次做流出报文处理，记录所用时间。统计 h_f 和 h_g 中冲突元素的数量。统计 h_f 和 h_g 量每个 hash 地址对应链表的长度的方差。在 h_f 中依次查找生成的<*a*, *b*>二元组，记录查找时间；在 h_g 中依次查找生成的<*d*, *e*，*c*, *a*, *b*>，记录查找时间。

（3）实验结果和分析

图 4-4 是三类执行时间的统计结果，图 4-5 是 hash 表元素冲突数量的统计结果，图 4-6 是 hash 表冲突链表长度方差的统计结果。

在图 4-6 中没有体现 PJWHash，是因为 PJWHash 的 h_f 元素链表长度方差是其他几种 hash 函数的数十倍，影响其他几种 hash 函数的对比效果。从实验

结果来看，在散列程度方面，除 DJBHash 和 PJWHash 以外，其他 3 种 hash 函数的散列程度相差不大。在计算速度方面，BPHash 略有优势。

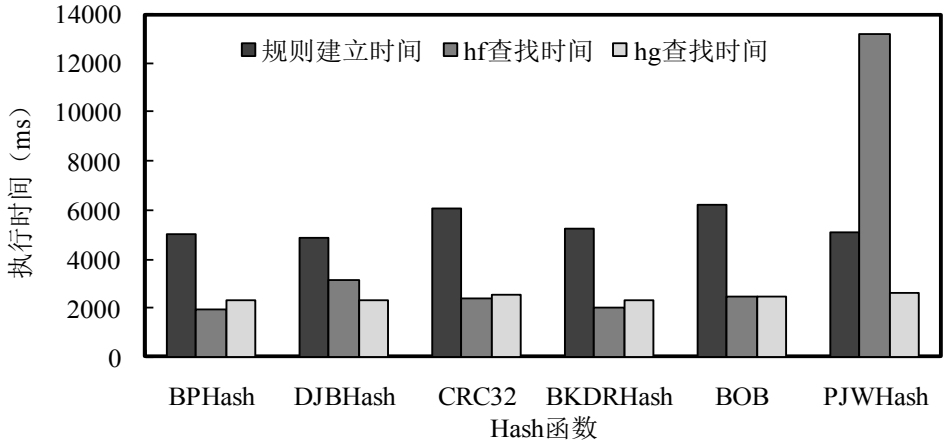


图 4-4 执行时间对比图

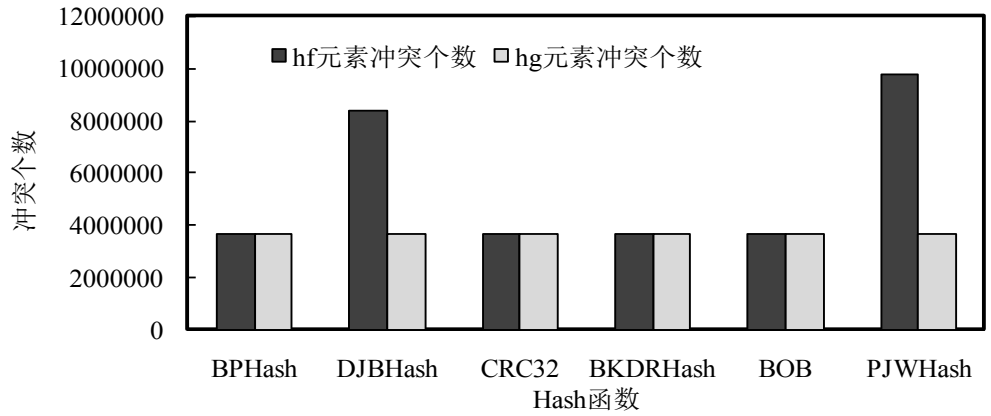


图 4-5 Hash 表元素冲突数量对比图

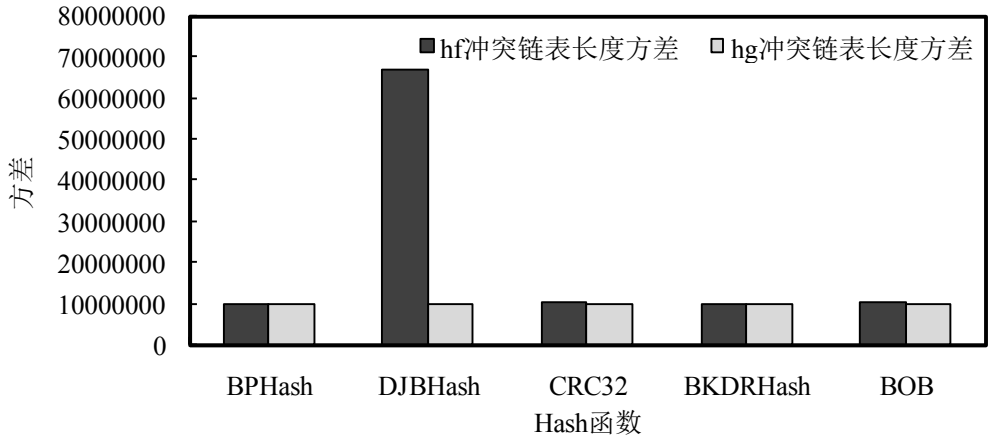


图 4-6 Hash 冲突链表长度方差对比图

从图 4-5 可以看出 DJBHash 和 PJWHash 对 h_f 的散列程度较差，分析可能是由于在 h_f 的源 IP、源端口六个字节的关键字中，虚拟地址 4 个字节差别较小造成的。将虚拟地址关键字的前两字节和后两字节分别加上源端口的值，构成新的关键字，再计算 hash 值，得到执行时间对比结果如图 4-7 所示，hash 表冲突元素数量对比如图 4-8 所示，hash 冲突链表长度方差如图 4-9 所示。

从图 4-7、4-8 和 4-9 中可以看出，在增加了 h_f 关键字的随机程度之后，DJBHash 和 PJWHash 对 h_f 的散列程度有明显提高，其他几种 hash 函数没有明显变化。BPHash 计算速度的优势由于增加了关键字处理变得更加微弱。另外，文献[49]和文献[50]，都指出 BOBHash 在计算速度方面更有优势，而本实验并没有得出该结果，推测可能是由于采用了文献[48]中 hashlittle 的实现方式，该实现方式较为复杂。

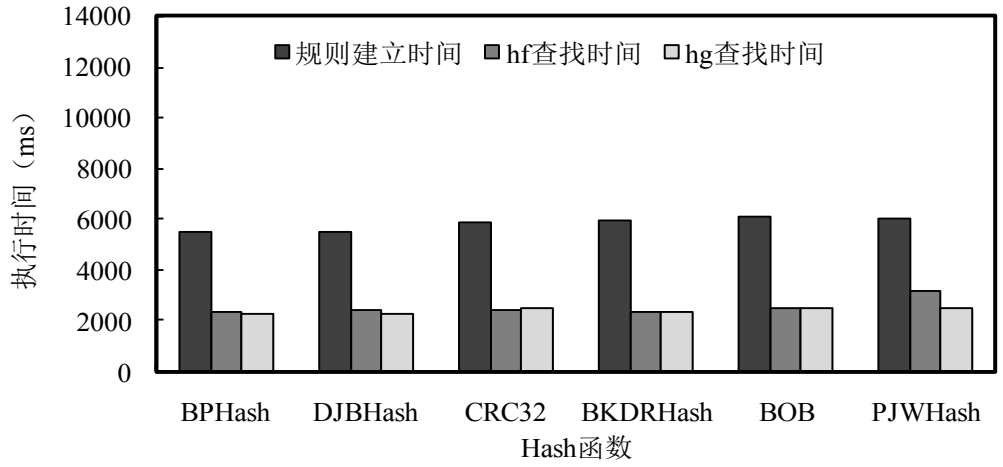


图 4-7 执行时间对比图

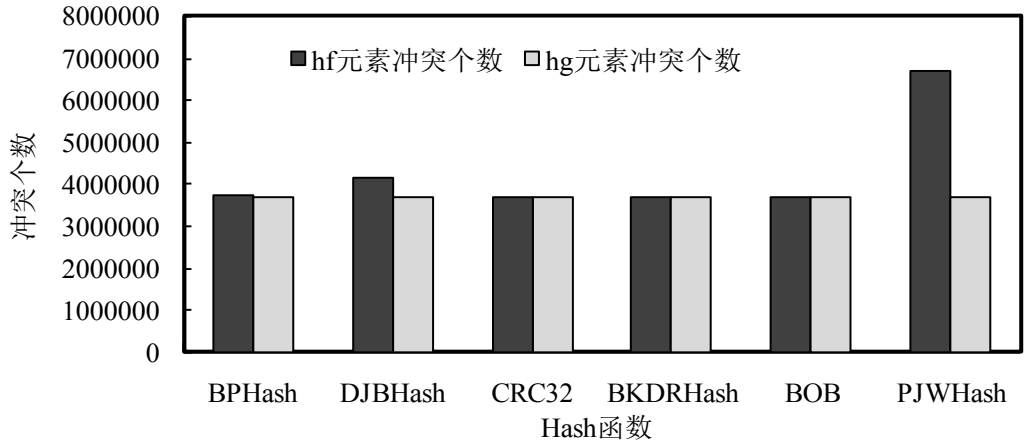


图 4-8 Hash 表元素冲突数量对比图

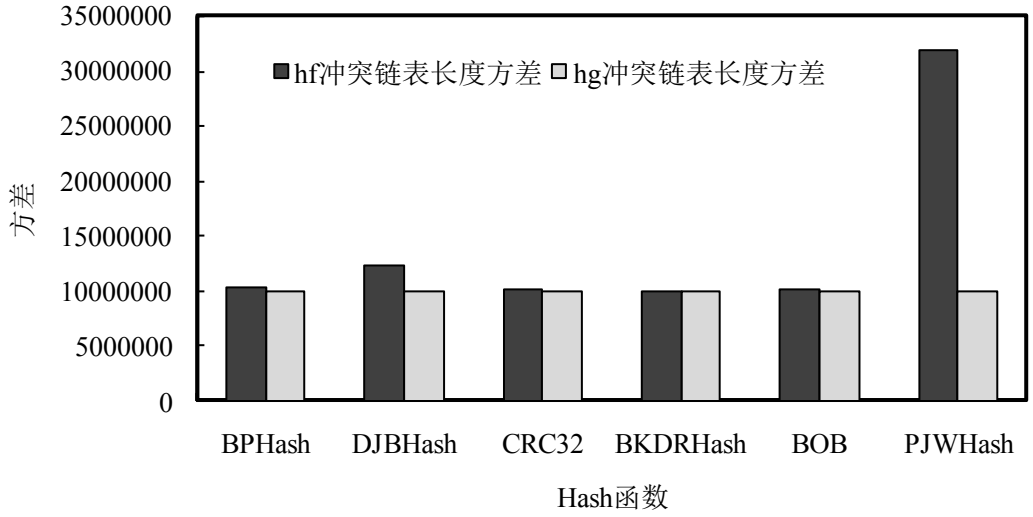


图 4-9 Hash 冲突链表长度方差对比图

综上所述，得出如下结论，第一，在特定的情况下，并不是所有的 hash 函数都有较好的散列程度和计算速度，比较 hash 函数的性能是有意义的；第二，关键字的随机程度会影响 hash 函数的散列程度，在 hash 表数据结构的设计上需要考虑这方面的影响；第三，在本文所介绍的 VPN 网关用户态 NAT 中，选择使用 BPHash。

4.4 用户态 NAT 测试和分析

本小节对前述用户态 NAT 进行性能测试。测试在实验室局域网内进行，使用 iperf 测试工具进行测试。测试设备的连接示意如图 4-10 所示。实验使用的设备参数如表 4-7 所示。

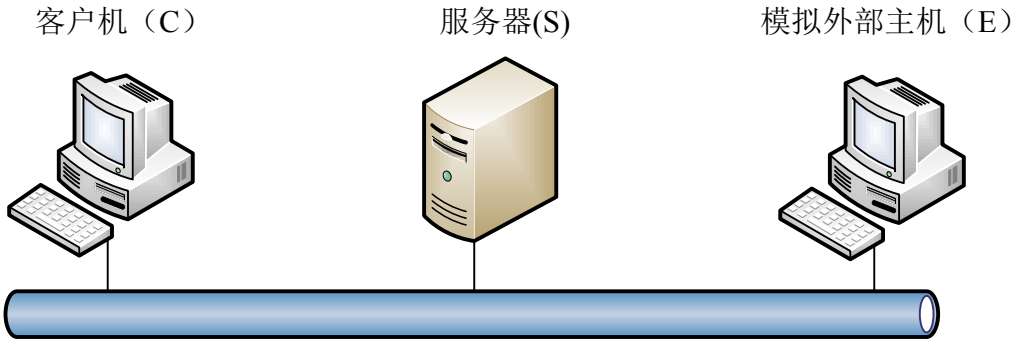


图 4-10 用户态 NAT 测试设备连接示意图

表 4-7 设备配置参数表

设备代号	配置参数	配置描述
S	CPU 型号	Intel(R) Atom(TM) CPU D525 @ 1.80GHz
	RAM 容量	1G
	网卡型号	Intel Corporation 82583V Gigabit Network Connection
	操作系统	CentOS Linux release 7.2.1511
C	CPU 型号	Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
	RAM 容量	4G
	网卡型号	Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
	操作系统	Ubuntu 14.04.2 LTS
E	CPU 型号	Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
	RAM 容量	4G
	网卡型号	Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
	操作系统	Ubuntu 14.04.4 LTS

环境配置：客户机 C 运行 VPN 网关客户端，服务器 S 运行 VPN 网关用户态 NAT，模拟外部主机 E 看做 VPN 外网主机。通过修改 VPN 网关代码，将网关下一跳 MAC 设置为 E 的 MAC。在 E 上进行 iptables 的 DNAT（目的地址转换）设置，使得目的地址是 10.10.0.1 的报文都会被修改成 E 自己的 IP。在 C 和 E 上安装 iperf 测试工具。这样，C 的 iperf 测试数据会先流经 S，解密后做 NAT 转换，由于将 S 的下一跳 MAC 改成了 E 的地址，故 NAT 后的报文将发往 E，E 在接收到报文后会先做 DNAT 转换，将目的地址 10.10.0.1 转换为 E 的 IP，然后交给 E 上运行的 iperf 服务端处理。

实验步骤：C 运行 VPN 客户端，与 S 建立 VPN 连接。在 E 中运行 iperf 服务端，在 C 中运行 iperf 客户端，C 中 iperf 的目标地址设为 10. 10.0.1。使用 iperf 默认 TCP 窗口大小测试 C 和 E 之间的 TCP 带宽。逐渐修改流出规则匹配条目在流出规则 hash 表的冲突链表中的位置，得到 TCP 带宽和匹配规则在流出规则 hash 表的冲突链表中的位置的关系如图 4-11 所示。

测试表明上文设计的用户态 NAT 实现了代理转发功能。从图 4-11 中可以看到当匹配规则是冲突链表的第一个结点时，带宽最大，位置越靠后，带宽越低，几乎呈线性关系，位置向后移动一位，带宽约下降 3Mb/s。匹配规则在冲突链表中的位置，代表找到该规则需要操作指针并比较关键字的次数，每多一次操作，都会影响到带宽。这说明 Hash 函数的选择是相当重要的，hash 函数

将关键字分布的越均匀，平均冲突链表就越短，平均带宽就越大。Hash 函数和关键字的选择仍是一个需要继续深入研究的课题。

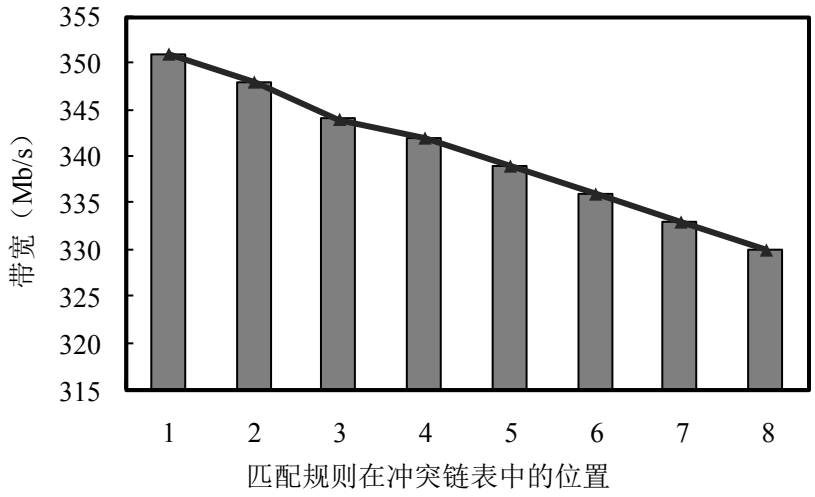


图 4-11 匹配规则在冲突链表中的位置和带宽的关系图

4.5 本章小结

本章介绍了 NAT 的基本情况、分类，介绍了 VPN 网关中用户态 NAT 和普通 NAT 的异同，给出了 NAT 的映射本质，给出了一种基于尝试的 NAT 算法，设计并实现了一种 VPN 网关用户态 NAT，该设计包括数据结构，流出报文处理，流入报文处理和 NAT 规则删除等模块，并通过实验为其选了 BPHash 作为其 hash 函数。最后测试了用户态 NAT 的性能。测试表明本文设计的用户态 NAT 实现了代理转发功能，并且 hash 函数和关键字的选择仍然是一个需要深入研究的课题。下一章将介绍基于 DPDK 的 VPN 网关整体的设计实现和测试分析。

第 5 章 原型系统的设计与测试分析

本章将在已有的研究基础之上，设计并实现基于 DPDK 的 VPN 网关（DPDK VPN Gateway，DVG）。然后，将 DVG 和其他五种软件 VPN 网关进行对比测试，分别测试了 VPN 网关连接转发功能和代理转发功能的性能。最后，对测试结果进行讨论和分析。

5.1 DVG 的设计

5.1.1 DVG 整体结构设计

DVG 的整体结构如图 5-1 所示。DVG 主要包括 DPDK 报文收发平台、用户态基础协议栈、Session 管理、虚拟 IP 管理、加密、VPN 路由查找、NAT 等模块。各模块说明如下。

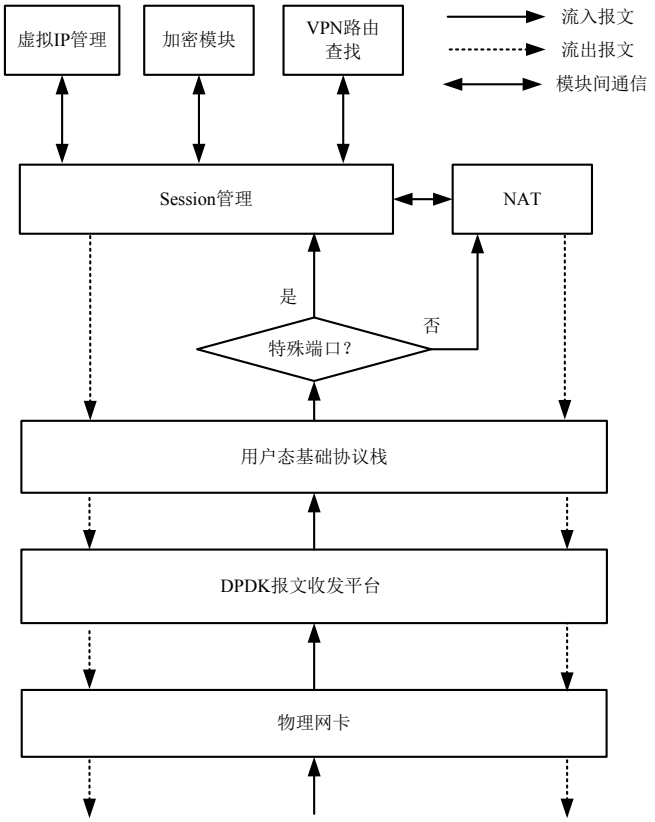


图 5-1 DVG 整体结构图

（1）DPDK 报文收发平台

高性能报文处理平台，负责报文接收和发送。使用轮询模式和用户态驱动收发报文。在第 2 章已经详细介绍，这里不再具体说明。

（2）用户态基础协议栈

一般网络使用的网络协议是 TCP/IP 协议族，所以基础协议指的是 TCP/IP 类协议，而基础协议栈实现的是 TCP/IP 协议族中的一个子集。该协议栈工作在用户态，绕开了内核协议栈，负责报文外层封装的解析、脱去和添加。为上层处理做准备，为发送报文做准备。

（3）Session 管理

Session 管理是网关中的一个关键模块。将一个客户端的一次连接称作一个 session，客户端可以多次连接或断开与 VPN 的连接，每次连接都是不同的 session。客户端一次连接所使用的地址信息、密钥信息都存放在 session 中。本模块负责管理 session，使用状态机控制 session 的生命周期，调用其他模块控制连接的建立、通信和销毁。

（4）虚拟 IP 管理

管理虚拟 IP 的存储、分配和回收。虚拟 IP 在内存中进行管理。虚拟 IP 统一分配 10.8.0.0/16 网络地址。使用二维数组维护虚拟 IP。二维数组中存放虚拟 IP 和是否可使用标志。使用一个整数索引加速查找可用虚拟 IP 的过程。初始时虚拟 IP 全部设置为可用，并将索引设为 0。当需要分配虚拟 IP 时，使用索引指向的 IP，将其是否可使用标志设置为已使用，然后将索引设置为下一个可用的虚拟 IP 在二维数组中的位置。当回收虚拟 IP 时，由虚拟 IP 网络字节序的后两个字节计算出该虚拟 IP 在二维数组中的位置，将是否可用标志置为可用。该模块较简单，下文不再详细说明。

（5）加密模块

本模块基于 Libsodium 实现了加解密及密钥生成和交换等认证加密相关功能。Libsodium 是一个先进而且易用的加密库。主要用于加密、解密、签名和生成密码哈希等。它是一个可移植的、跨编译器支持、可安装的加密库，提供兼容 API。本文采用 Libsodium 支持的 XSalsa20Poly1305 作为本网关的认证加密算法。加密并不作为本文重点，故下文不再详细说明。

（6）VPN 路由查找

根据内层报文的虚拟 IP 查找实际地址和密钥，采用基于划分阶段即 Patricia 树和 hash 表结合的方式实现。利用 session 管理模块来记录实际地址和密钥等

连接信息，故在实现中本模块并未再次存储实际地址和密钥，而是将 session 信息使用 Patricia 树和 hash 表重新组织。其他内容已在第 3 章详细介绍，本章不再详细说明。

（7）NAT 模块

内层报文通过网关访问外网时需要调用该模块。该模块负责对报文进行源地址或目的地址转换，并记录转换规则。该模块的实现在第 4 章中已详细介绍，本章不再具体说明。

5.1.2 用户态基础协议栈设计

基础协议栈在用户态实现。它实现的是网络协议中的一个子集。该子集至少包含能够完成连接转发功能和代理转发功能所需要的基础网络协议。由于目前一般网络通常使用 TCP/IP 协议族，故本文讨论的基础协议栈是 TCP/IP 协议族的一个子集。至少需要实现的协议如表 5-1 所示。

表 5-1 用户态基础协议栈协议表

协议所在网络层次	协议名称
传输层	UDP、TCP（MSS）
网络层	IPv4
链路层	ARP、Ethernet II

链路层至少需要实现 ARP 和 Ethernet II 协议。VPN 网关需要对 ARP 协议作出响应，否则，客户端无法与 VPN 网关通信，这是客户端连接 VPN 网关的基础。Ethernet 协议用于对链路层的报文的封装和解封装，是解析上层协议的基础。

网络层至少需要实现 IPv4 协议。目前 IP 层协议中 IPv4 和 IPv6 并行使用，但大部分网络都支持 IPv4，所以至少实现 IPv4 协议的解析，用于上层协议的解析和 IP 地址的获取。

在传输层协议中，对于 UDP 协议，由于 VPN 外层封装使用 UDP 协议，故必须实现 UDP 协议。对于 TCP 协议， VPN 网关并不直接处理，但是需要控制内层负载中的最大报文段长度（Maximum Segment Size，MSS）字段。MSS 是指 TCP 报文负载的最大长度。控制 MSS 字段可以避免 TCP 报文分片，降低网关处理复杂度。控制 MSS 字段值放在客户端进行，减轻网关处理压力。控制 MSS 字段的方法是，在客户端发送报文前判断报文类型，如果发送报文是 TCP

报文，并且 SYN 标识为 1，则将其 MSS 选项的值修改为需要的值，例如 1400，然后重新计算 TCP 校验和并填充 TCP 报文头部的 TCP 校验和字段，再发送修改后的报文。

5.1.3 Session 管理模块设计

Session 管理是 VPN 网关中的一个重要模块。用于识别 VPN 连接，控制 VPN 连接的状态，处理 VPN 隧道内部数据流的解析。一个 VPN 连接的创建、通信和销毁都由该模块控制，是整个 VPN 功能的关键模块。

（1）Session 数据结构设计

将一个 VPN 客户端的一次连接称作一个 session，每个 session 有独立的密钥信息。VPN 网关接收到的有效报文由外层封装和内层加密负载组成，解密负载需要找到对应 session。这就需要保证一个 session 的所有报文都有排他的特征。这种特征有两种，第一种是现有的，可以利用外层封装的源地址，即源 IP 和源端口二元组；第二种是人为添加的，可以在负载中添加明文的标识符来区分每个 session。第一种特征不需要添加新的参数，可以降低处理的复杂度，但外层封装的源地址可能由于网络变化而产生变化。第二种特征不依赖于外层封装，有较强的稳定性，但为每个报文添加特征信息会增加处理消耗，并且分配和维护统一的全局特征会带来一定的复杂度。考虑到外层封装的稳定性较高，即使发生变化也可以引入重连协议来更新 session，本文采用第一种方式来作为特征。

经过上述分析，本文使用 hash 表来存储 session 信息。Hash 表元素的结构如表 5-2 所示。

表 5-2 Session 元素结构表

类别	中文描述	字母代号
关键字	外层封装源 IP	src_IP
	外层封装源端口	src_port
非关键字	密钥信息	secret
	虚拟 IP	v_IP
	到达网关前一跳物理地址	ether_addr

关键字是外层封装的源地址，由于客户端到达 VPN 网关可能会经过 NAT，该源地址可能不是客户端实际的源地址，而是 VPN 网关解析到的源地址。密钥

信息用于加解密，其中存放着 session 的密钥等用于加解密的必须信息。虚拟 IP 中存放 VPN 网关为客户端分配虚拟 IP，也就是内层负载中的报文所使用的 IP。到达网关前一跳物理地址用于当 VPN 网关需要向该客户端发送报文时，填充链路层封装。

Session 表的冲突元素采用链表的形式链接在一起。为减少报文处理过程中空间分配所消耗的时间，在创建 hash 表时，维护一块空闲空间，使用链表链接在一起，当有空闲空间时，直接使用空闲空间，当空闲空间不足时，才进行申请空间操作。Session 管理还包含 session 表的添加、查找和删除。这三种操作都是 hash 表的基本操作，不再赘述。Session 表的添加、查找和删除的时机受接入控制模块的控制，将在接入控制模块中说明。

（2）Session 生命周期管理

Session 管理内部采用了状态机机制，包括公钥交换（Key Exchange，KE）、用户认证（User Authentication，UA）、加密通信（Encrypted Communication，EC）和断开连接（Destroy Connection，DC）四个阶段。由客户端报文驱动完成四个阶段的切换。客户端和 VPN 网关在 session 四个阶段的交互过程可以用图 5-2 表示。四个阶段的交互过程描述如下。

第一阶段是公钥交换阶段，客户端向网关发送客户端公钥，网关接收到报文后对报文外层封装进行解析，在 session 表中查找解析出的源地址。由于这是客户端和网关的第一次通信，所以 session 表的查询结果为空。此时，网关判断该 session 处于第一阶段。网关生成公钥和私钥对，在 session 表中插入一条 session 元素，将路由信息和密钥信息存入 session 元素中，将其状态置为 EX。然后向客户端发送服务端公钥。

第二阶段是用户认证阶段，客户端接收到网关公钥后，生成通信密钥材料，使用客户端私钥、服务端公钥和密钥材料生成本次连接的通信密钥。然后向网关发送该密钥材料和加密后的客户端口令。网关本次 session 查找将查找到上次存储的 session，并由客户端发送的参数和 session 的 EX 状态，判断该 session 进入第二阶段 UA。网关使用私钥、客户端公钥和客户端的密钥材料生成 session 的通信密钥，该密钥由认证加密算法保证和客户端生成的保持一致。网关使用生成的密钥解密出用户口令，验证该口令。如果验证失败，则删除该 session，向客户端发送验证失败。如果验证成功，则分配一个可用的虚拟 IP，将 session 状态更新为 UA，并向客户端发送虚拟 IP。

第三阶段是数据通信阶段，客户端用户认证成功后就与网关建立了连接，

就可以与其他客户端通信或使用网关访问外部网络了。此时客户端向网关发送加密数据。网关查找到该 session 后，由状态标志为 UA 或 EC 判断其处于第三阶段，使用 session 中的密钥解密出该内层负载。解析出内层报文的目的地址，进行 VPN 路由查找，如果查找到了某个 session，判断为连接模式，使用该 session 的密钥重新加密负载，并使用该 session 中的实际地址添加外层封装，转发该报文。如果未找到路由，则判断为 NAT 转发模式，进行 NAT 处理后转发该报文。最后，如果状态标志为 UA 则将其更新为 EC。

第四阶段是断开连接阶段，客户端向网关发送加密的断开口令。网关查找到该 session 后，由状态标志 EC 和客户端发送参数判断客户端希望断开连接。解密负载后得到用户口令，验证该口令，如果验证通过，则进行断开处理，回收虚拟 IP，删除 session。如果未验证通过，则丢弃该报文。

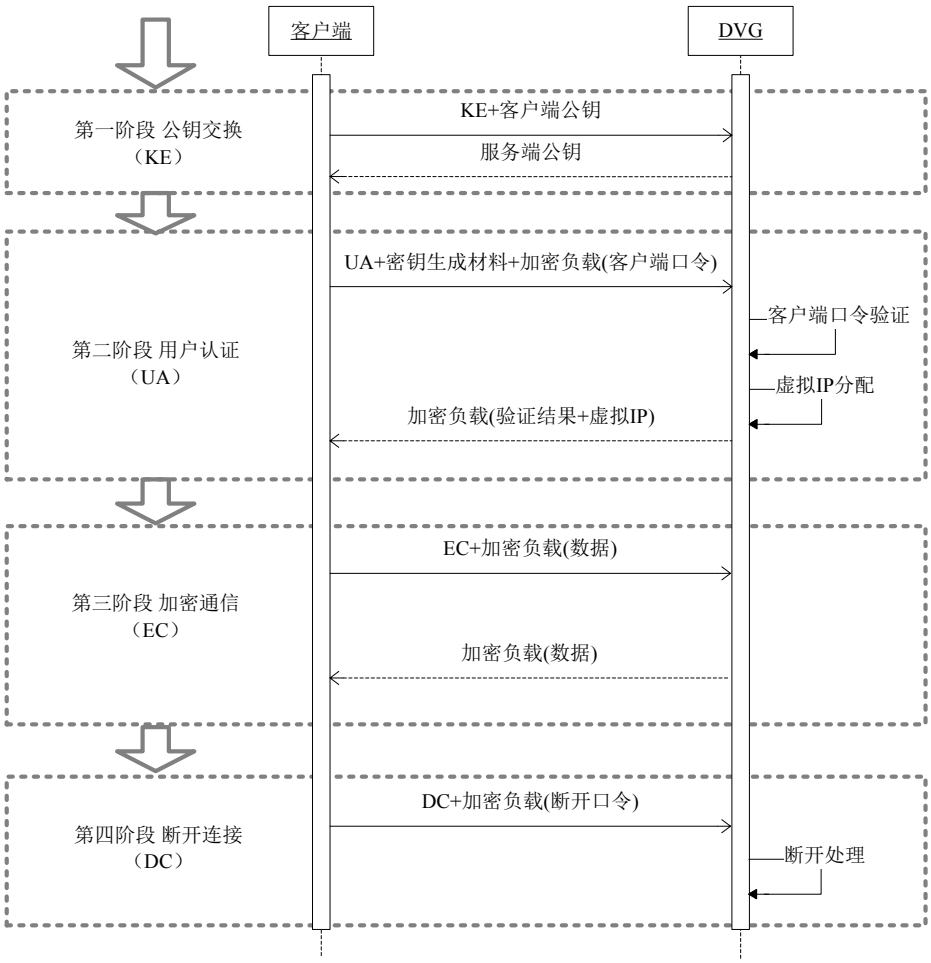


图 5-2 Session 生命周期示意图

5.2 连接转发功能测试

5.2.1 测试环境

测试设备的连接示意如图 5-3 所示。两台客户机和服务器都连接在实验室同一网络地址的局域网中。测试设备的配置参数如表 5-3 所示。

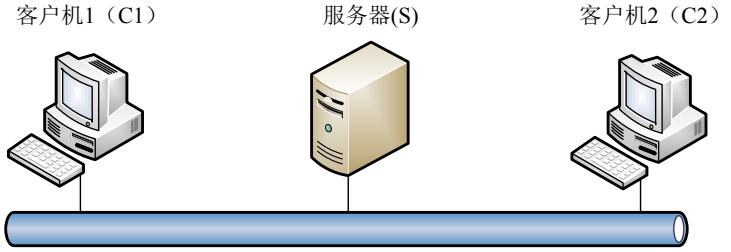


图 5-3 测试设备连接示意图

表 5-3 设备配置参数表

设备名称	配置参数	配置描述
服务器	CPU 型号	Intel(R) Atom(TM) CPU D525 @ 1.80GHz
	RAM 容量	1G
	网卡型号	Intel Corporation 82583V Gigabit Network Connection
	操作系统	CentOS Linux release 7.2.1511
客户机 1	CPU 型号	Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
	RAM 容量	4G
	网卡型号	Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
	操作系统	Ubuntu 14.04.4 LTS
客户机 2	CPU 型号	Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
	RAM 容量	4G
	网卡型号	Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
	操作系统	Ubuntu 14.04.2 LTS

5.2.2 测试对象

测试对象是六种软件 VPN 网关，这六种网关的说明如下：

（1）DVG：本文描述的基于 DPDK 的高性能 VPN 网关。

（2）VG：和本文描述的协议相同但使用内核做报文处理。

（3）openvpn-gw：一种使用 SSL 协议的软件 VPN 网关。

（4）ipsec-gw：一种运行 IPSec 协议的软件 VPN 网关。

（5）accel-ppp-gw：一种高性能 PPTP 软件 VPN 网关。

（6）pptp-gw：一种运行 PPTP 协议的软件 VPN 网关。

服务器 S 安装的各网关软件版本如表 5-4 所示。

表 5-4 六种软件 VPN 网关说明表

网关名称	软件版本	配置
DVG	--	XSalsa20Poly1305 加密算法，PAP 用户认证
VG	--	XSalsa20Poly1305 加密算法，PAP 用户认证
openvpn-gw	OpenVPN 2.3.9	Blowfish 加密算法，X.509 证书认证
ipsec-gw	strongSwan U5.4.0	AES-18-CBC 加密算法，X.509 证书认证
accel-ppp-gw	ACCEL-PPP 1.8.0	MPPE 加密算法，MSCHAP 协议认证
pptp-gw	pptpd v1.4.0	MPPE 加密算法，MSCHAP 协议认证

5.2.3 测试工具

测试使用 iperf 作为测试工具。iperf 是一种强大的网络测试工具，通过调整参数，可以较准确地测出网关的 UDP 和 TCP 性能。C1 运行 iperf 的服务端，是报文的接收者；C2 运行 iperf 的客户端，是报文的发送者。C1 安装的 iperf 版本是 2.0.5，C2 安装的 iperf 版本是 2.0.8b。

Iperf UDP 性能测试使用的命令如表 5-5 所示。

表 5-5 Iperf UDP 性能测试命令表

设备	iperf 测试命令
C1	iperf –u –s –i 1
C2	iperf –u –c ip –b bw –l len –t tm

在 C2 的命令中，ip 表示 C1 的虚拟 IP；bw 表示测试带宽，是发送者发送数据达到的带宽；len 表示发送报文的长度；tm 表示测试时间。

Iperf TCP 性能测试使用的命令如表 5-6 所示。

表 5-6 Iperf TCP 性能测试命令表

设备	iperf 测试命令
C1	iperf –s –i 1
C2	iperf –c ip –M 1400 –t tm

在 C2 的命令中，ip 表示 C1 的虚拟 IP；将-M 参数设置为 1400，表示 TCP 的最大负载长度是 1400；tm 表示测试时间。

5.2.4 测试过程

连接转发功能分别测试六种网关的 UDP 性能、TCP 性能、时延和 jitter。每种软件 VPN 网关的测试过程如下：

（1）建立 VPN 连接：服务器运行 VPN 网关，C1 和 C2 运行网关客户端，通过 VPN 网关建立起 VPN 连接。

（2）UDP 性能测试：C2 使用 iperf 向 C1 发送 UDP 数据包。C2 将 iperf 命令中的-l 参数 len 分别设置为 64、128、256、768、1024、1280、1400。在这 7 种 iperf 测试包长下调整 iperf 的-b 参数 bw，使得网关刚好不丢包，稳定一段时间后，记录此时的吞吐量、包转发率和 jitter。

（3）TCP 性能测试：TCP 性能测试包括 iperf TCP 带宽测试和文件下载速率测试。iperf TCP 吞吐量测试：C2 使用 iperf 向 C1 发送 TCP 数据，使用默认 83.5KB 作为初始窗口大小。稳定一段时间后，记录 TCP 实际吞吐量。文件下载速率测试：在 C1 上部署 Apache 服务，并放置一个大小为 1.85G 的文件。C2 使用 wget 命令下载 C1 上的该文件。下载完成之后记录平均下载速率。

（4）时延测试：C1 使用 ping 命令向 C2 发送 10 个报文，记录每个报文的往返时延（Round-Trip Time，RTT）和最后的平均 rtt。

5.3 代理转发功能测试

代理转发功能测试的测试环境使用和连接转发功能测试相同的环境和设备，此处不再说明。

测试对象是五种软件 VPN 网关，分别是 DVG、openvpn-gw、ipsec-gw、accel-ppp-gw 和 pptp-gw 不包括 VG，因为该网关并未实现代理转发功能。这五种网关的说明和配置已在连接转发功能测试中描述，此处不再赘述。

使用 iperf 作为测试工具，iperf 工具的命令和使用和连接转发功能测试相同，此处不再赘述。

代理转发功能分别测试五种网关的 UDP 性能、TCP 性能和 jitter，不测试时延。因为 DVG 的用户态 NAT 未实现 ICMP 协议，故无法使用 ping 命令测试时延。

代理转发功能测试需要先配置 S 和 C2。配置 C2 的方法是：在 C2 上设置一条 iptables 的 DNAT 规则，遇到目的地址为 10.10.0.1 的报文，就将其目的地址修改为 C2 的 IP 地址。配置 S 的方法是：当测试 DVG 时，需要通过修改代码将 S 下一跳 MAC 设置为 C2 的地址。当测试其他网关时，需要在 S 上配置一条路由，将发往 10.10.0.0/16 的报文转发到 C2，并在 S 上配置一条 iptables 的 SNAT 规则，VPN 网络内层报文所使用的虚拟源地址改为 S 的地址。

剩余测试过程除未测试时延外，与连接转发功能测试相同。

5.4 测试结果讨论和分析

5.4.1 连接转发功能测试结果和分析

（1）UDP 性能测试

不同包长下的 UDP 吞吐量测试结果如图 5-4 所示。从该图中可以看出，DVG 的 UDP 吞吐量明显高于其他五种网关，约是性能最低的 pptp-gw 的 7.03 到 13.04 倍，包长越小优势越明显。整体上，六种网关吞吐量都随 iperf 测试包长逐渐增加，并逐渐趋于平缓，但 accel-ppp-gw 和 ipsec 在 iperf 测试包长为 1400 时的吞吐量低于测试包长为 1280 时的吞吐量。

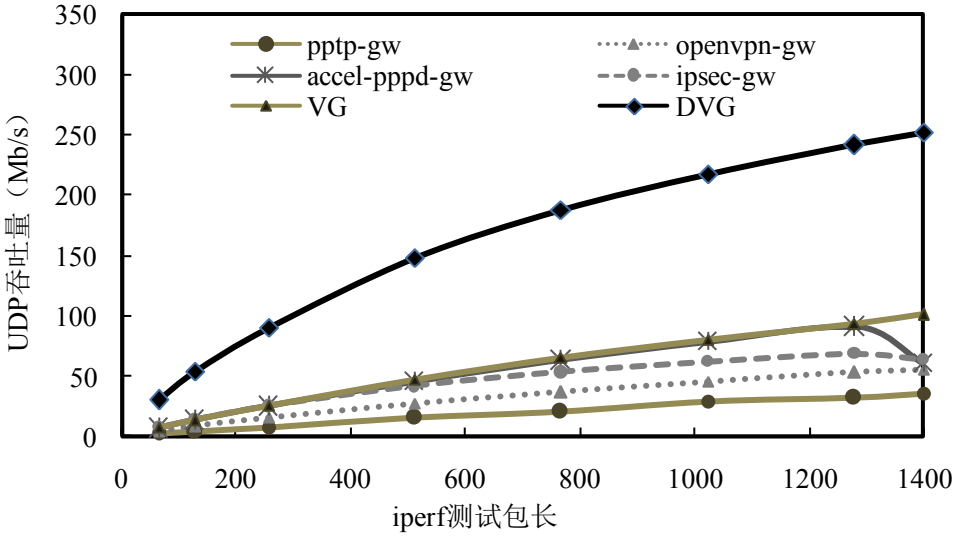


图 5-4 不同包长下吞吐量对比图

不同包长下网关恰好不丢包时的包转发率测试结果如图 5-5 所示。从图中可以看出，DVG 的包转发率明显高于其他五种网关，约是性能最低的 pptp-gw

的 7.02 到 13.04 倍，测试包长越小，优势越明显。整体上，六种网关的包转发率都呈下降趋势。

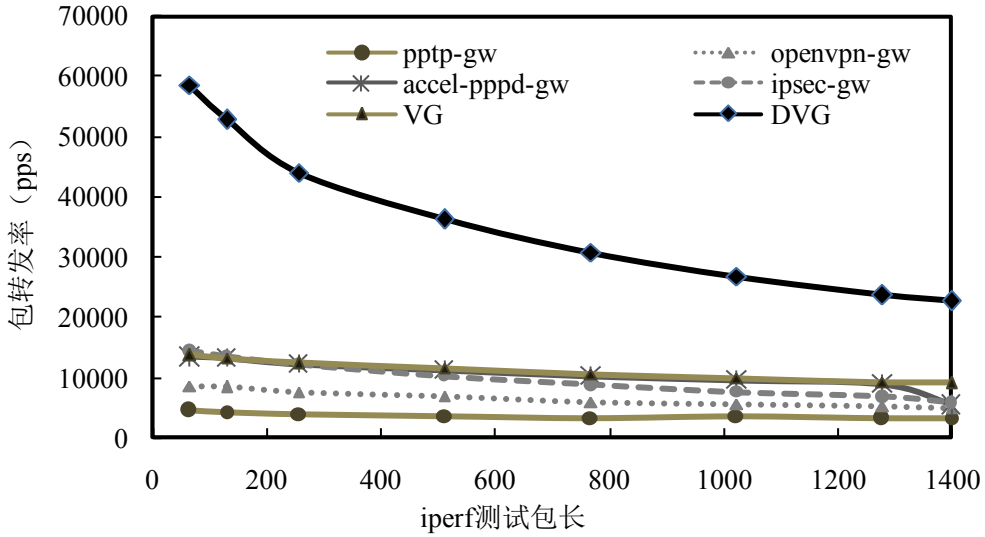


图 5-5 不同包长的包转发率对比图

（2）TCP 性能测试

iperf TCP 吞吐量测试结果如图 5-6 所示。文件下载速率测试结果如图 5-7 所示。由于目前大部分应用都基于 TCP 协议，故 VPN 网关的 TCP 性能是一个重要参数。从图中可以看出，两种 TCP 性能测试结果大体相似。DVG 的性能远高于其他五种网关，约是性能最低的 pptp-gw 的 5 倍。在实际测试中，发现 pptp-gw 的稳定性较差，测试 TCP 吞吐量和文件下载速率时，波动较大，并有一定周期性规律。

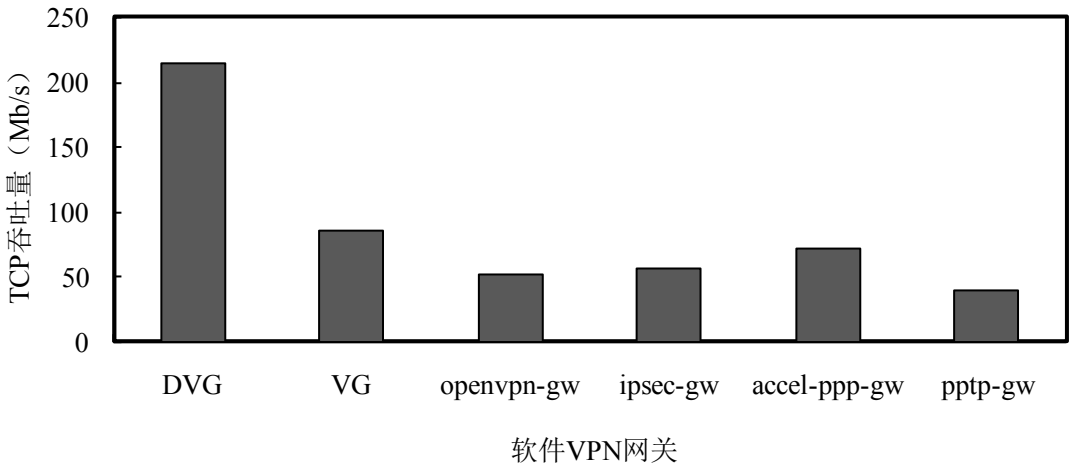


图 5-6 TCP 吞吐量对比图

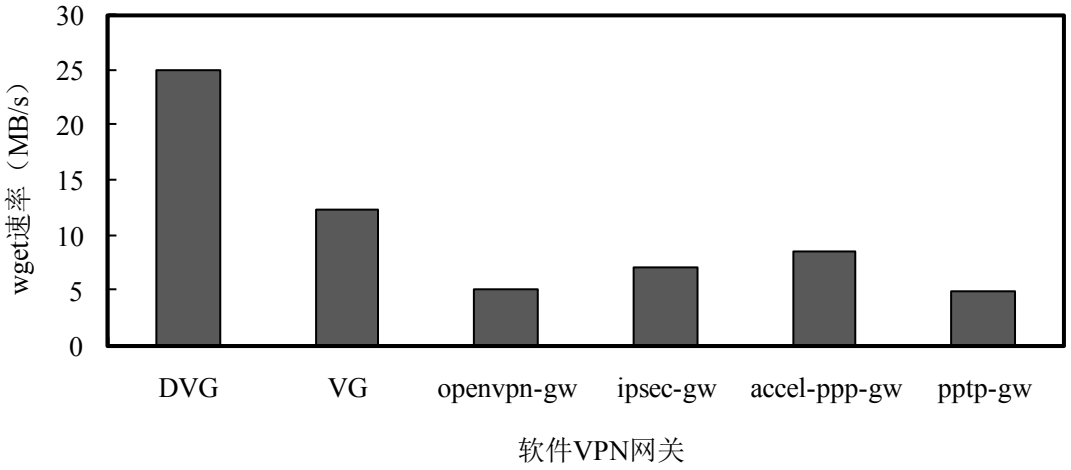


图 5-7 文件下载速率对比图

（3）时延测试

时延测试结果如图 5-8 所示。从图中可以看出，DVG 在 10 次测试中，每次的时延都是六种网关中最低的，有较优的时延特性。

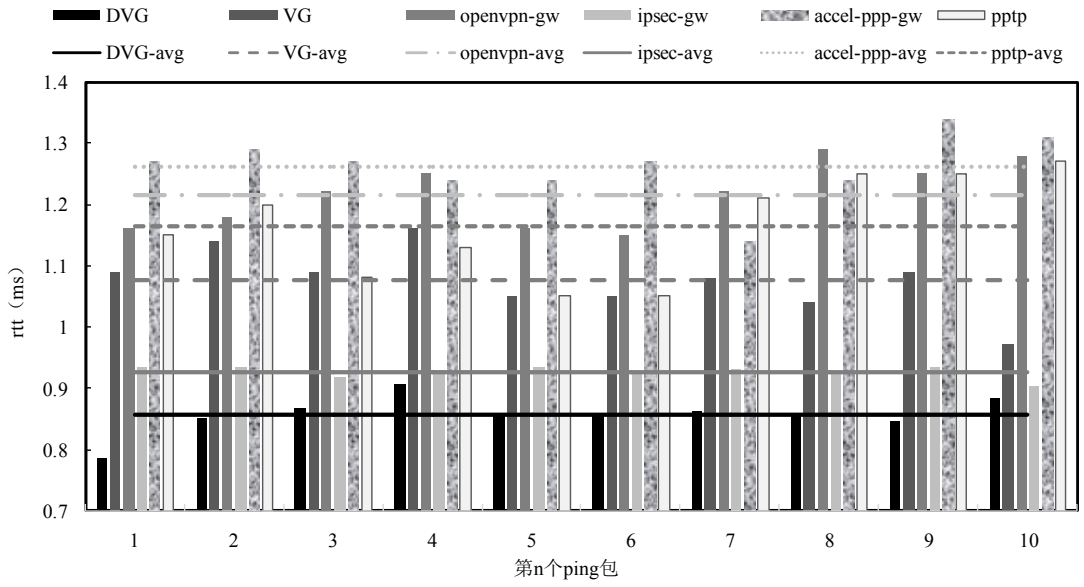


图 5-8 时延对比图

（4）Jitter 测试

Jitter 测试结果如图 5-9 所示。从图中可以看出，DVG 和 pptp-gw 的 jitter 最低，说明这两种网关较为稳定。但结合前面的测试，DVG 的 jitter 较低说明 DVG 的高性能较为稳定，而 pptp-gw 的 jitter 值较低，说明其性能维持在一个

较低的水平。整体上，jitter 呈上升或平缓趋势，但 openvpn-gw 较为反常，在包长较小时不稳定而在包长较大时稳定性更强。

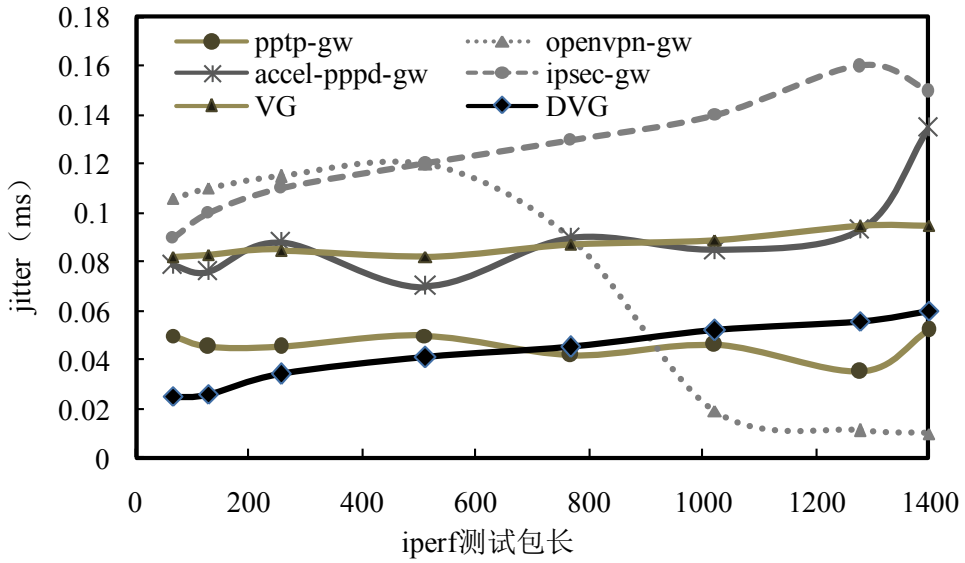


图 5-9 jitter 对比图

5.4.2 代理转发功能测试结果和分析

（1）UDP 性能测试

不同包长下 UDP 吞吐量测试和包转发率测试结果如图 5-10、图 5-11 所示。

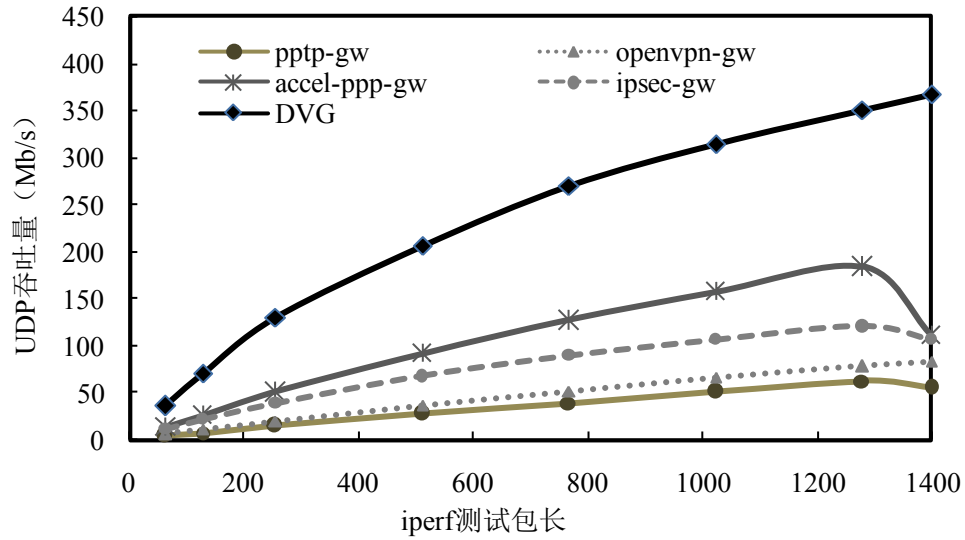


图 5-10 不同包长下吞吐量对比图

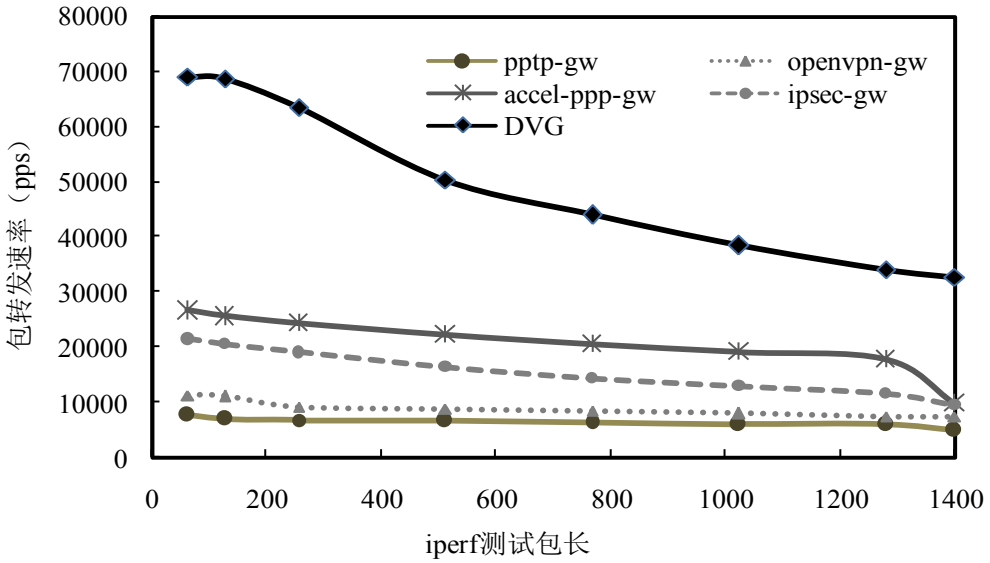


图 5-11 不同包长的包转发率对比图

从图 5-10 和图 5-11 中可以看出，DVG 的 UDP 吞吐量和包转发率明显高于其他四种网关，约是性能最低的 pptp-gw 的 5.65 到 9.93 倍，包长越小优势越明显。整体上，五种网关吞吐量都随 iperf 测试包长逐渐增加，并逐渐趋于平缓，但 pptp、accel-ppp-gw 和 ipsec 在 iperf 测试包长为 1400 时的吞吐量低于测试包长为 1280 时的吞吐量。五种网关的包转发率都呈下降趋势。

（2）TCP 性能测试

iperf TCP 吞吐量测试结果如图 5-12 所示。文件下载速率测试结果如图 5-13 所示。从图中可以看出，两种 TCP 性能测试结果大体相似。DVG 的性能远高于其他四种网关，约是性能最低的 openvpn-gw 的 5 到 6 倍，约是 pptp-gw 的 3.7 倍到 4.6 倍，约是 ipsec-gw 的 3 倍。实际测试中 pptp-gw 和 openvpn-gw 波动较大，其他三种网关 TCP 性能较为稳定。

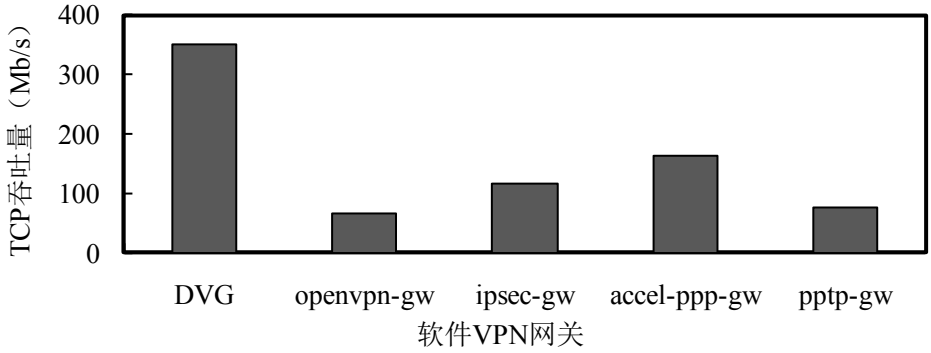


图 5-12 TCP 吞吐量对比图

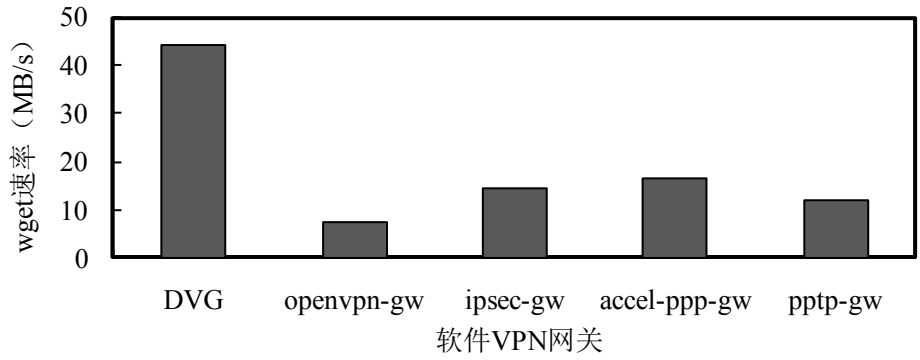


图 5-13 文件下载速率对比图

（3）Jitter 测试

Jitter 测试结果如图 5-14 所示。从图中可以看出，DVG 的 jitter 最低，并且曲线相对平稳，说明 DVG 较为稳定。openvpn-gw 和 accel-pppd-gw 稳定性随包长变化比较明显，pptp-gw 在小包长和大包长下 jitter 较低，其他包长下 jitter 值较为稳定。

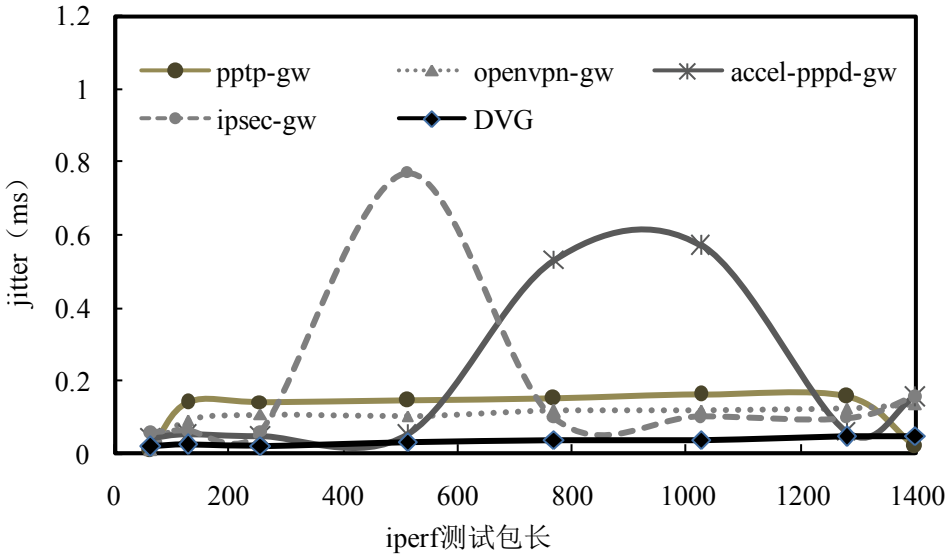


图 5-14 jitter 对比图

5.4.3 整体测试结果讨论和分析

在连接转发功能测试结果中，DVG 的性能明显优于其他五种软件 VPN 网关。将性能最低的 pptp-gw 的性能作为标准，设为 1，除去参数 jitter，可以将六种网关性能的倍率关系做成表 5-7。在代理转发功能测试结果中，DVG 的性能明显优于其他四种软件 VPN 网关。同样将 pptp-gw 性能作为标准，设为 1，

除去参数 jitter，可以将五种网关代理转发性能的倍率关系做成表 5-8。

表 5-7 六种网关连接转发性能倍率关系表

参数	pptp	openvpn	ipsec	accel-ppp	VG	DVG
UDP 吞吐量	1	1.54-1.91	1.76-3.17	1.67-2.96	2.80-3.04	7.03-13.04
包转发率	1	1.54-1.91	1.76-3.19	1.67-2.96	2.80-3.04	7.02-13.04
TCP 吞吐量	1	1.31	1.42	1.79	2.17	5.35
文件下载速率	1	1.03	1.43	1.73	2.49	5.1
rtt	1	1.04	0.79	1.09	0.84	0.74

表 5-8 五种网关代理转发性能倍率关系表

参数	pptp	openvpn	ipsec	accel-ppp	DVG
UDP 吞吐量	1	1.26-1.61	1.95-2.99	2.02-3.70	5.65-9.93
包转发率	1	1.26-1.61	1.95-2.99	2.02-3.70	5.65-9.93
TCP 吞吐量	1	0.9	1.53	2.16	4.62
文件下载速率	1	0.62	1.21	1.38	3.66

由于不同包长下的倍率关系不同，故倍率关系在一个区间中。从表 5-7 和表 5-8 中可以看到，DVG 的性能远高于其他几种软件 VPN 网关。并且稳定性最强。从原始数据中发现，报文长度越短，DVG 的优势就越明显。

表中没有列出 jitter 参数，是因为 jitter 参数变化较为随机，没有多个网关没有明显倍率关系。但从两个功能的 jitter 结果看，DVG 的曲线都在最下方，表示在各个包长下的时延波动都较小，较为稳定。

从表 5-7 中可以看出，VG 的性能排在第二位。VG 采用了和 DVG 相同的协议，但使用了传统 VPN 网关的框架，没有采用高性能报文平台加用户态协议栈的模式。它性能较高的原因推测是加密算法速度较快，并且采用了轻量级的协议。只观察 DVG 的性能和 VG 的性能，可以排除加密和协议的影响，可以看到，DVG 的 UDP 性能约是 VG 的 2.51 到 4.29 倍，TCP 性能约是 VG 的 2 倍多。说明高性能报文平台加用户态协议栈模式在 VPN 网关中起到了良好效果。并且，测试的包转发率的最大值约在 60000pps，速率较低，在速率更高的情况下，内核报文处理的性能瓶颈将更加突出。

然而，DVG 也有缺点。由于本文描述的 DPDK 报文处理使用了轮询模式，所以 DVG 的 CPU 占用率始终是 100%，即使在无报文经过的情况下，DVG 的

CPU 占用率也在 100%。所以，DVG 的最佳应用场景是报文长期处于接近满负荷的情况。但 DPDK 也可以使用非轮询模式，可以引入一种切换方式，当流量较小时，采用非轮询模式处理报文，当流量超过某个阈值，就采用轮询模式，以此较少消耗。

5.5 本章小结

本章介绍了课题原型系统 DVG 的整体结构、主要模块和两种主要功能的设计和实现。对比测试了六种软件 VPN 网关的连接转发功能和代理转发功能的 UDP、TCP、时延和 jitter 等性能，测试结果表明 DVG 的各项性能均优于其他五种网关，并且包长越小优势越明显。

结　论

传统软件 VPN 网关使用内核进行网络处理，效率低。本文致力于提升软件 VPN 网关吞吐量、包转发率和时延等性能的方法，主要完成了以下工作：

（1）总结了 DPDK 加速技术，设计了基于 DPDK 的 VPN 网关的框架。该框架使用用户态驱动，利用轮询模式收发报文，构建精简的用户态协议栈，并在用户态实现 VPN 网关的连接转发功能和代理转发功能。

（2)优化了基于 Patricia 树的路由查找算法，使之更适用于 VPN 路由查找。在此基础上，结合 VPN 路由查找的特点，提出一种基于划分阶段的改进方法。实验表明，在少量用户分散于大量网络地址的条件下基于 Patricia 树的 VPN 路由查找算法灵活性强、查找速度快；在大量用户集中少量网络地址的条件下，基于划分阶段的改进方法效率更高。

（3）研究了用户态 NAT，使用映射描述了 NAT 技术核心，给出一种 NAT 核心算法，根据该算法，设计了一种用户态 NAT 实现方法。通过实验为该设计选择了 BPHash 作为 hash 函数。通过测试验证了该设计能够实现用户态 NAT 功能。测试了匹配规则在 hash 表的冲突链表中的位置和 TCP 带宽的关系，在该测试环境下，匹配规则在链表中向后移动一位，带宽下降约 3Mb/s。

（4）设计实现了一种基于 DPDK 的软件 VPN 网关（DVG）。在实验室局域网环境下，与其他五种软件 VPN 网关进行了对比测试。测试结果表明，DVG 的吞吐量、包转发率、时延和时延抖动等性能均优于其他五种软件 VPN 网关，并且包长越小优势越明显。在该测试环境下，连接转发功能的性能最大可达 PPTP 网关的 13 倍，IPSec 网关的 4.1 倍，SSL 网关的 4.2 倍；代理转发功能的性能最大可达 PPTP 网关的 9.9 倍，IPSec 网关的 3.3 倍，SSL 网关的 6.2 倍。

本文设计了基于 DPDK 的高性能 VPN 网关框架，研究了 VPN 路由查找和用户态 NAT，实现了基于 DPDK 的高性能软件 VPN 网关。根据研究、实现和实验中遇到的问题，未来可以从以下几个方面继续开展工作：

（1）多核框架在基于 DPDK 的软件 VPN 网关中的应用。多核框架并不是单线程框架的简单叠加，需要解决路由信息、密钥信息等资源共享的问题。未来打算从三方面入手，第一，采用锁机制；第二，引入 CPU 核间通信机制；第三，将信息分布在 CPU 核上，采用分阶段处理，每个阶段流量重新分流到对应 CPU 核中。

（2）并发性能的提升。并发性能也是一个较为重要的性能评价指标，未来打算从三个方面入手，第一，并发性测试的标准；第二，并发性测试的方法；第三，提升并发性的结构和方法。

参考文献

[1] Han K, Yang D, Liu J J. The Design of Secure Embedded VPN Gateway[C]//2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA). IEEE, 2014: 350-353.

[2] Yu D, Chen N, Tan C. Design and Implementation of Mobile Secu rity Access System (MSAS) Based on SSL VPN[C]// International Workshop on Education Technology and Computer Science. IEEE, 2009: 152-155.

[3] Kilinc C, Booth T, Andersson K, et al. WallDroid: Cloud Assisted Virtualized Application Specific Firewalls for the Android OS[C]//Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE Computer Society, 2012: 877-883.

[4] 傅春乐. Android 移动终端 Wi-Fi 安全接入关键技术的研究与实现[D]. 哈尔滨：哈尔滨工业大学学位论文，2016：52-64.

[5] Shue C, Shin Y, Gupta M, et al. Analysis of IPSec Overheads for VPN Servers[C]//1st IEEE ICNP Workshop on Secure Network Protocols, 2005. (NPSec). IEEE, 2005: 25-30.

[6] Shue C A, Gupta M, Myers S A. IPSec: Performance analysis and enhancements[C]//2007 IEEE International Conference on Communications. IEEE, 2007: 1527-1532.

[7] 单蓉胜，李建华，王明政. 安全网关优化[J]. 小型微型计算机系统，2005，26（5）：753-756.

[8] Narayan S, Fitzgerald M, Ram S. Empirical Network Performance Evaluation of IPSec Algorithms on Windows Operating Systems Implemented on a Test-bed[C]//2010 IEEE International Conference on Computational Intelligence and Computing Research. IEEE, 2010: 409-412.

[9] Padhiar S, Verma P. A Survey on Performance Evaluation of VPN on various Operating System[J]. International Journal of Engineering Development and Research, 2015, 3(4): 516-519.

[10] Redžović H, Smiljanić A, Savić B. Performance Evaluation of Software Routers with VPN Features[C]//24th Telecommunications forum TELFOR 2016. IEEE, 2016: 1-4.

[11] Lawas J B R, Vivero A C, Sharma A. Network Performance Evaluation of VPN Protocols (SSTP and IKEv2)[C]//2016 Thirteenth International Conference on

Wireless and Optical Communications Networks(WOCN). IEEE, 2016: 1-5.

[12] 梅松，李之棠. 一种新的高性能 VPN 系统的模型分析[J]. 小型微型计算机系统，2006，27（5）：793-797.

[13] 李湘峰，赵有健，全成斌. 对称密钥加密算法在 IPsec 协议中的应用[J]. 电子测量与仪器学报，2014，28（1）：75-83.

[14] Turan F, Clercq R, Maene P, et al. Hardware Acceleration of a Software-based VPN[C]//2016 26th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2016: 1-9.

[15] 刘磊，孔志印，赵荣彩，汪伦伟. 基于多核的 IPSec 并行处理技术研究与实现[J]. 计算机工程，2009，35（3）：139-141.

[16] 周振斌，唐剑琪. 基于负载均衡的高吞吐量 IPSec VPN 系统[J]. 计算机工程与应用，2012，48（36）：85-89.

[17] 刘振钧，李治辉，林山. 基于 FPGA 的万兆网的 IPsec ESP 协议设计与实现[J]. 通信技术，2015，48（2）：242-246.

[18] Ros-Giralt J, Commike A, Honey D, et al. High-Performance Many-Core Networking: Design and Implementation[C]// INDIS '15: Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science. ACM, 2015: 1-7.

[19] R R, Ramia K B, Kulkarni M. Integration of LwIP stack over Intel DPDK for high throughput packet delivery to applications[C]//2014 Fifth International Symposium on Electronic System Design. IEEE, 2014: 130-134.

[20] 王佰玲，方滨兴，云晓春. 零拷贝报文捕获平台的研究与实现[J]. 计算机学报，2005，28（1）：46-52.

[21] Asai H. Where are the Bottlenecks in Software Packet Processing and Forwarding?[C]//CFI '14 Proceedings of The Ninth International Conference on Future Internet Technologies. ACM, 2014: 1-6.

[22] Liao G D, Zhu X, Bhuyan L. A New Server I/O Architecture for High Speed Networks[C]//2011 IEEE 17th International Symposium on High Performance Computer Architecture. IEEE, 2011: 255-256.

[23] Gallenmüller S, Emmerich P, Wohlfart F, et al. Comparison of Frameworks for High-Performance Packet IO[C]//2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2015: 29-38.

[24] Linux Foundation Project. DPDK[EB/OL]. [2017-05-21]. http://dpdk.org/.

[25] Rizzo L. Netmap: a novel framework for fast packet I/O[C]//Proceedings of

USENIX ATC Conferences. USENIX, 2012: 101-112.

[26] Maria Teresa Allegro. PF_RING ZC (Zero Copy)[EB/OL]. [2017-05-21]. http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/.

[27] Raumer D,Gallenmüller S, Emmerich P, et al. Efficient serving of VPN endpoints on COTS server hardware[C]//2016 5th IEEE International Conference on Cloud Networking. IEEE, 2016: 164-169.

[28] Ajayan A C, P P, Krishnan R, et al. Hiper-Ping: Data Plane Based High Performance Packet Generation Bypassing Kernel on x86 based Commodity Systems[C]//2016 Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2016: 478-483.

[29] Xu Z J, Zhou L, Feng L, et al. PacketUsher: a DPDK-Based Packet I/O Engine for Commodity PC[C]//2016 IEEE/CIC International Conference on Communications in China (ICCC). IEEE, 2016: 1-5.

[30] Zhang W, Liu G, Zhang W, et al. OpenNetVM: A Platform for High Performance Network Service Chains[C]// HotMIddlebox '16: Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization. ACM, 2016: 26-31.

[31] 何佳伟，江舟. 基于 Intel DPDK 的高性能网络安全审计方案设计[J]. 网络与信息工程，2016，（Z1）：87-91.

[32] Redžović H, Vesović M, Smiljanić A, et al. Energy-efficient network processing based on netmap framework[J]. ELECTRONICS LETTERS, 2017, 53(6): 407-409.

[33] Kim J H, Na J C. A Study on One-way Communication using PF_RING ZC[C]//2017 19th International Conference on Advanced Communication Technology (ICACT). IEEE, 2017: 301-304.

[34] Han S J, Jang K, Park K S, et al. PacketShader: A GPU-Accelerated Software Router[C]//Proceedings of the ACM SIGCOMM 2010 conference. ACM, 2010: 195-206.

[35] Honda M, Huici F, Raiciu C, et al. Rekindling Network Protocol Innovation with User-Level Stacks[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(2): 53-58.

[36] TANG Lu, YAN JinLi, SUN ZhiGang, et al. Towards high-performance packet processing on commodity multi-cores: current issues and future directions[J]. Science China(Information Sciences), 2015, 12(12): 28-43.

[37] Li B, Tan K, Luo L, et al. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware[C]// SIGCOMM '16:

Proceedings of the 2016 ACM SIGCOMM Conference. ACM, 2016: 1-14.

[38] Shim J, Kim J, Lee K, et al. Knapp: A Packet Processing Framework for Manycore Accelerators[C]//2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era. IEEE, 2017: 57-64.

[39] Coonjah I, Catherine P C, Soyjaudah K M S. Performance Evaluation and Analysis of Layer 3 Tunneling between OpenSSH and OpenVPN in a Wide Area Network Environment[C]//2015 International Conference on Computing, Communication and Security (ICCCS). IEEE, 2015: 1-4.

[40] Barylski M. On IPSec Performance Testing of IPv4/IPv6 IPSec Gateway[C]//Proceedings of the 2008 1st International Conference on Information Technology. IEEE, 2008: 1-4.

[41] Ely D, Savage S, Wetherall D. Alpine: A User-Level Infrastructure for Network Protocol Development[C]// USITS'01 Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems. USENIX, 2001: 1-13.

[42] 刘风华，丁贺龙，张永平. 关于 NAT 技术的研究与应用[J]. 计算机工程与设计，2006，27（10）：1814-1817.

[43] Gamess E, Morales N. Implementing IPv6 at Central University of Venezuela[C]//LANC '07: Proceedings of the 4th international IFIP/ACM Latin American conference on Networking. ACM, 2007: 43-51.

[44] Srisuresh P, Holdrege M. IP Network Address Translator (NAT) Terminology and Considerations[S]. USA: IETF, 1999-08. https://www.ietf.org/html/rfc2663.txt.

[45] Rosenberg J, Weinberger J, Huitema C, et al. STUN-Simple Traversal User Datagram Protocol (UDP) Through Network Address Translators (NATs)[S]. USA: IETF, 2003-03. https://www.ietf.org/html/rfc3489.txt.

[46] Partow A. General Purpose Hash Function Algorithms[EB/OL]. [2017-05-21].http://www.partow.net/programming/hashfunctions/index.html.

[47] K. Lovett. Miscellaneous hash functions[EB/OL]. [2017-05-21]. http://www.call-with-current-continuation.org/eggs/hashes.html.

[48] Jenkins B. lookup3.c[EB/OL]. [2017-05-21]. http://www.burtleburtle.net/bob/c/lookup3.c.

[49] Henke C, Schmoll C, Zseby T. Empirical Evaluation of Hash Functions for Multipoint Measurements[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 41-50.

[50] Molina M, Niccolini S, Duffield N G. A Comparative Experimental Study of

Hash Functions Applied to Packet Sampling[C]//Performance Challenges for Efficient Next Generation Networks. Beijing University of Posts and Telecommunications Press, 2008: 39-48.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于 DPDK 的高性能 VPN 网关的研究与实现》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：穆瑞超　　日期：2017 年 6 月 16 日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1)学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：穆瑞超　　日期：2017 年 6 月 16 日

导师签名：　　日期：2017 年 6 月 16 日

致　谢

时光飞逝，两年的硕士学习生活转眼就要结束。在此，对教导我的老师、一起努力的同学和朋友们表示深深的感谢，是你们让这两年时光充实快乐。

感谢我的导师佟晓筠教授。佟老师治学严谨，对工作认真严格，对学生和蔼可亲、关怀备至，是一位让人敬佩的长者。佟老师为本文的撰写给予了耐心细致的指导，使我有很大的提高。

感谢王佰玲教授。王老师学识渊博，思维敏锐，不仅教会我很多科研经验和方法，开拓了我的视野和思路，还教会我很多做人做事的道理。本文的主要研究工作是在王老师的指导下完成的，在论文定稿之际，向王老师表示诚挚的感谢。

感谢傅春乐学长。研一开始就在傅学长的带领下做工程，傅学长思维活跃，科研能力强，让我在学习和生活中受益良多，是我学习的楷模，并为本文的完成提出了很多宝贵意见。感谢孙云霄学长。孙学长作为项目带头人，在工程实践上经验丰富、视野开阔、技术非凡。当我在项目中遇到困难时，孙学长总能给出解决的思路，多次手把手指导，让我的工程能力、调试能力有很大提高。

感谢网络技术研究所团队的所有老师和同学。感谢刘扬老师对本文修改提出的宝贵意见。感谢辛国栋老师在工程项目上的指导。感谢唐宏彬老师在实验室日常生活中的关心。感谢郭帅君同学帮助我整理测试数据。感谢并肩努力的周光凯同学、张新星同学、吴昊同学。

感谢我的室友季志宇同学、潘恒康同学、梁毅同学。两年来相互交流、共同努力，一起度过了这段充实愉悦的时光。

感谢我的亲人们，尤其是我的父母，感谢他们多年来的的支持和鼓励，今后会更加努力，用更好的成绩来回报他们的养育之恩。

最后，诚挚地感谢论文评阅人以及各位专家教授对本文提出的宝贵意见。