哈爾濱工業大學

毕业设计（论文）

题　　目　基于 DPDK 的高性能 IPSec VPN 的研究与实现

专　　业　信息安全

学　　号　150120526

学　　生　殷　悦

指 导 教 师　刘　扬

答 辩 日 期　2019 年 6 月 13 日

摘　要

随着互联网的普及，网络流量急剧增加，为了提供更好更安全的服务，分布于全国各地的企业通常使用专线，但铺设专线价格不菲，因此企业使用公网传输逐渐得到普及。为了保证数据的安全，IDC 通常使用通过 IPSec VPN 的方式在互联网上传输数据。使用传统协议栈的 IPSec VPN 在高速的网络中，性能已经达到了瓶颈，用户态协议栈可以较好的提升网络的性能，DPDK 是英特尔公司开发的一款优秀的用户态协议栈平台，本文使用 DPDK 对 IPSec VPN 的性能提升展开研究。

首先，为了解决传统协议栈的数据处理低效问题，本文从传统协议栈和用户态协议栈展开研究。传统的协议栈在数据处理的过程中存在上下文切换频繁，数据复制冗余，对多核心支持不佳，锁竞争开销大等问题。本文介绍了 DPDK 使用大内存页、用户空间 IO、处理器亲和性等技术来决解这些问题。

为了实现 IPSec 数据包的处理，本文实现了数据包的加解密，认证、拆包与封装等功能。由于 DPDK 没有使用内核协议栈，需要用户基于 DPDK 的接口和框架实现所需的协议栈。本文参考 RFC 和内核协议栈进行实现。

为了解决用户态和内核态之间的数据通信，本文研究了 KNI 网卡技术。由于 IPSec 通信中 IKE 为密钥协商数据包，特点是通信数据量少，而实现复杂，因此需要借助传统协议栈来完成。ESP 为通信数据包，特点是通信数据量大，实现简单，需要使用 DPDK 进行加速。

为在用户态实现代理转发功能，本文研究了网络地址转换技术，并给出一种 NAT 算法。该方法使用两个哈希表记录 NAT 规则，实现了流出数据的 SNAT 和流入数据的 DNAT，并实现了 NAT 规则的超时清理功能，并重新计算数据包的新校验和。

为了解决数据链路层网络中物理地址问题，本文实现了网络地址解析协议算法和以太网协议头封装。通过向目的主机发送 ARP 请求，解析返回的 MAC 地址，并存储在哈希表中。当发送数据包时，根据 ARP 映射表，为其封装上以太网头。

最后，基于上述研究基础，本文设计并实现了基于 DPDK 的高性能 IPSec VPN，并与使用传统协议栈的 IPSec VPN 进行了性能对比测试。测试结果表明本文基于 DPDK 实现的 IPSec VPN 在数据包的吞吐量，包转发率，延迟方面均优于传统协议栈的 IPSec VPN，当数据包长度越小，优势越明显。

关键词：IPSec；VPN；DPDK；用户态协议栈

Abstract

With the popularity of the Internet, network traffic has increased dramatically. In order to provide better and safer services, enterprises distributed throughout the country usually use dedicated lines, but the laying of dedicated lines is expensive, so the use of public network transmission has gradually gained popularity. In order to ensure data security, IDC usually uses IPSec VPN to transmit data over the Internet. IPSec VPN using traditional protocol stack has reached the bottleneck in high-speed network. The user-mode protocol stack can improve the performance of the network. DPDK is an excellent user-state protocol stack platform developed by Intel Corporation. DPDK conducts research on the performance improvement of IPSec VPN.

Firstly, in order to solve the problem of data processing inefficiency of traditional protocol stack, this paper studies from the traditional protocol stack and user-state protocol stack. In the process of data processing, the traditional protocol stack has frequent context switching, redundant data replication, poor support for multiple cores, and large competition for locks. This paper introduces DPDK to use large memory pages, user space IO, processor affinity and other technologies to solve these problems.

In order to realize the processing of IPSec data packets, this paper implements the functions of encryption, decryption, authentication, unpacking and encapsulation of data packets. Since the DPDK does not use the kernel protocol stack, the user needs to implement the required protocol stack based on the DPDK interface and framework. This article is implemented with reference to the RFC and the kernel protocol stack.

In order to solve the data communication between user mode and kernel mode, this paper studies KNI network card technology. Since IKE is a key negotiation data packet in IPSec communication, it is characterized by a small amount of communication data and a complicated implementation, and therefore needs to be completed by using a conventional protocol stack. ESP is a communication data packet, which is characterized by a large amount of communication data and simple implementation, and needs to be accelerated by using DPDK.

In order to implement the proxy forwarding function in the user mode, this paper studies the network address translation technology and gives a NAT algorithm. The method uses two hash tables to record the NAT rules, implements the SNAT of the outgoing data and the DNAT of the incoming data, and implements the timeout cleaning function of the NAT rule, and recalculates the new packet. Checksum.

In order to solve the physical address problem in the data link layer network, this paper implements the ARP protocol algorithm and the encapsulation of the Ethernet

protocol header. The returned MAC address is parsed by sending an ARP request to the destination host and stored in the hash table. When a packet is sent, an Ethernet header is encapsulated according to the ARP mapping table.

Finally, based on the above research, this paper designs and implements a high performance IPSec VPN based on DPDK, and compares it with IPSec VPN using traditional protocol stack. The test shows that the IPSec VPN implemented in this paper is superior to the traditional protocol stack IPSec VPN in terms of packet throughput, packet forwarding rate and delay. When the packet length is smaller, the advantage is more obvious.

Keywords: IPSec, VPN, DPDK, user-mode protocol stack

目　录

第 1 章 绪　论

1.1 课题背景

随着网络通信技术的发展，大数据到来，用户和网络都日益增长。而保证网络的高速和安全问题成了重中之重。近几年来，网络安全事故频发，信息的传输受到了严重威胁。通常企业业务遍布全国各地，为了保障数据安全，可以使用专线进行业务通信，但铺设专线价格不菲，因此企业公网传输逐渐得到普及。为了保证数据的安全，IDC 通常使用通过 IPSec VPN 的方式在互联网上传输数据。

虚拟专用网络(Virtual Private Network)，即 VPN 使用了加密和隧道技术在公共网络中建立了一条虚拟专用的链路，使物理位置相距很远的人通过 VPN 技术也可以像在局域网中一样通信[1]。

常见的 VPN 技术有 PPTP／L2TP 和 IPSec，PPTP 与 L2TP 位于链路层，两种协议均基于 PPP 协议来封装数据包。PPTP 与 L2TP 的区别在于前者仅支持两端点间建立隧道，而后者可以可以支持两端点间建立多条隧道，且后者支持隧道模式下认证。而 IPSec 为目前最流行的 VPN 协议，IPSec 包含了一套英特网秘钥交换协议、认证头协议、封装安全载荷协议[2]。

IPv6 是下一代互联网安全协议，IPSec 作为 IP 层的安全协议，兼容了 IPv4 和 IPv6,为 IP 层数据报文提供了完整性、机密性、抗重放攻击、报文原地址认证。IPSec 可运行于任何支持 IP 协议的设备。作为 IPv6 的一重要组成部分，所有 IPv6 节点均需支持并使用 IPSec 协议，而 IPv4 可使用 IPSec,但并非必须使用 IPSec。除了 IPv4 和 IPv6 本身协议的不同，IPSec 协议在 IPv4 和 IPv6 的作用、功能、结构均相同[3]。

IPSec 由三部分组成：认证头协议(AH)，封装安全载荷协议(ESP)和英特网秘钥交换协议(IKE)[4]。认证头协议(AH)用于对报文源地址认证和报文完整性检测功能。封装安全载荷协议(ESP)用于报文内容认证和加密的功能，加密算法常用 AES、DES、3DES 等，完整性校验算法常用 HMAC-SHA1、HMAC-MD5 等。英特网秘钥交换协议(IKE)用于协商源主机和目的主机间用作保护 IP 报文的 ESP 和 AH 等参数，例如加密秘钥、秘钥生存周期、认证算法、加密算法等。IPSec 仅指 AH 和 ESP，IKE 使用 UDP 的 500 端口，是应用层协议。

1.2 研究的目的和意义

随着 VPN 通信规模增大，传统协议栈的 VPN 在性能上无法满足用户的需求。传统的 VPN 吞吐量低，包转发率低，时延大。协议栈集成于系统内核中，传统协议栈中断频繁，内存间复制多，功能冗余等缺点。通过构建用户态协议栈的 VPN 来弥补这些缺点，极大提高 VPN 性能，满足高速和高效等性能需求，因此开发一款安全高性能的 VPN 意义重大[5]。

1.3 国内外研究现状

近年来，各国重点大学和科研机构纷纷寻找用户状态协议栈加速的相关技术并获得了许多创新成果。

华盛顿大学开发了一款 FreeBSD 用户态协议栈 Alpine，避免了已有的软件迁移到该协议栈时还要进行太多更改的难题。Alpine 将内核协议栈虚拟为可以共享内核的用户空间协议栈。Alpine 消减了收发数据包所需的时间，使在用户态中开发协议栈更容易[6]。

华盛顿大学用户状态协议栈的其他研发为协议栈框架。网络接口、多线程和操作系统层则是这个框架的重点。网络接口用作数据流收发和传输 API。操作系统则管理修改其状况，如 TCP 状况或路由表消息；多线程库用作发送和接收数据流[7]。

wattcp 是加拿大 Erick Engelke 在 DOS 下开发的用户模式协议栈。wattcp 在协议栈中完成了传输层与网络层的数据管理，包含传输层与网络层包重组件碎片之间的交换；除此之外，避免了移植应用到新协议栈的麻烦，该协议栈还完成了 BSD 套件[8]。

OpenOnLoad[9]是一个来自 solarflare 的高性能网络协议栈。不但 OpenOnLoad 显著地减弱了时延和 CPU 负载，与此同时显著地提升了带宽和数据效率。OpenOnLoad 协议栈在 Linux 中使用，重现了规范的 BSD Socket API，用于直接地传输到 TCP/IP 协议栈中。 用户的网络程序可以在 OpenOnLoad 堆栈上运行而不需要修改。数据传输路径上直接避开了操作系统内核，使 OpenOnLoad 在用户空间中网络包处理能力的显著提高。但也不削弱安全性，可以使用传统协议栈的所有功能。

清华大学在中国研发了一套用户态空间协议栈 RCP。在数据通信过程中，RCP 避开内核并允许主机用户与用户态空间通信。RCP 协议和 NIC 设备间路径减少了网络包传输到用户态空间的频率，吞吐量也提高了不少。同时，通过 RCP 中的 API，定制编程非常简单。RCP 可以进行 PVM 并行开发，经过细微的改变，原始的 PVM 软件可以与 RCP 进行交互[10]。

1.4 本文研究内容及组织结构

本文研究利用 IPSec 通信协议进行通信。使用 IKE 协议进行证书认证及协商会话秘钥，对流经的数据使用 ESP 协议进行加密，使用 DPDK 平台解决了通信数据包处理低效问题。第二章介绍了传统协议栈存在的瓶颈和解决瓶颈的方法，并介绍了 DPDK 对数据包加密加速的方案。然后简单介绍了 IPSec 的原理和优势。第三章描述了 IPSec 网关总体设计方案，介绍了如何获取密钥，数据包流向。第四章简单介绍了整体的框架和每个部分所处的位置。第五章详细讲解了 ESP 数据包，ARP 数据包的构造和处理过程，讲解了 SADB 和 SPDB 的作用和实现，讲解了网络地址转换算法，校验和等算法。最后讲解了本程序环境配置和测试，并对测试结果进行分析。

第 2 章 IPSec 和 DPDK 框架的研究

本章首先介绍了 IPSec 几种协议的特点和原理，并针对 ESP 协议和 AH 协议展开研究。然后讨论了密钥交换 IKE 的几种模式、原理和特点。最后研究了传统协议栈面临的问题以及 DPDK 用户态协议栈的应对措施。

2.1 IPSec 综述

2.1.1 IPSec 的组成

由于 IP 协议设计之初没有安全保护，数据在传输中有可能被监听或篡改，存在安全隐患。而 IPSec 是一套完整的加密系统，IPSec 协议提供每个 IP 数据包的数据认证，从而保证传输中未被篡改。提供了数据包加密，从而保证数据机密性。

根据 RFC4301 中定义，IPSec 技术主要由 AH，ESP，IKE，ISAKMP/Oakley，加密算法及 SA 组成[11]。虽然 AH 协议有完整性校验功能，但不能加密数据。而 ESP 协议不但有完整性校验功能，还有加密功能。IKE 协议主用于秘钥管理，完成设备间会话秘钥协商和交换工作。当 IPSec VPN 建立连接时就需要确定加密和认证的算法。根据 RFC3602 定义 IPSec 加密常用 AES、3DES 等算法[12]。根据 RFC2401 定义，IPSec 认证常用 SHA-1 和 MD5 算法[13]。而 SA 协议用于在不同设备间算法协商和秘钥匙交换。

2.1.2 封装安全载荷协议与认证头协议的区别和作用

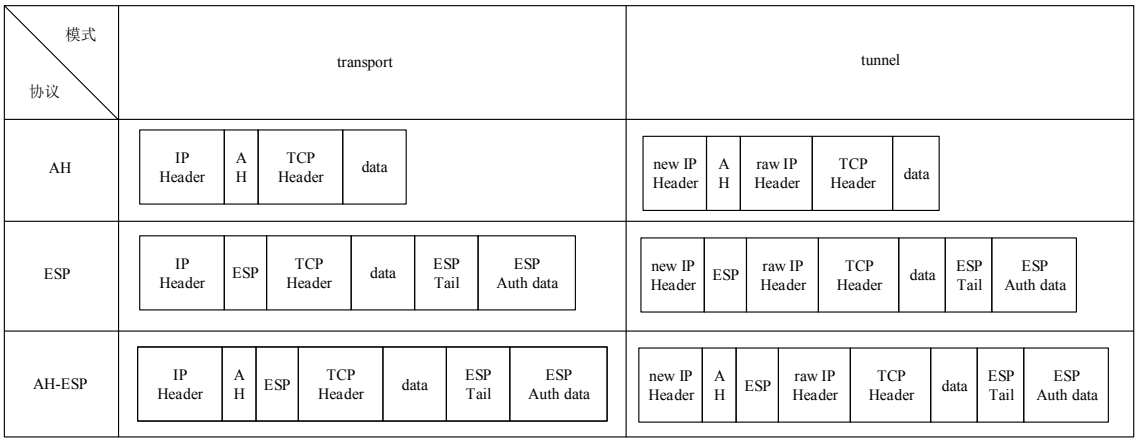

图 2-1 AH 和 ESP 数据包构造

根据 RFC4303 定义，ESP 协议提供了数据包的机密性，完整性认证和数据包的抗重播性，防止黑客监听数据包内容，篡改数据包，或进行重放攻击[14]。而 AH 协议只能保证数据包的完整性，防止黑客截断数据包或向数据包中插入伪造数据。ESP 包和 AH 包构造如图 2-1 所示。

2.1.3 Internet 密钥交换协议的组成和原理

IKE(Internet 密钥交换协议)由 ISAKMP，SKEME，Oakley 三部分组成。ISAKMP 使用了 UDP 的 500 端口，定义了信息交换的体系结构。SKEME 实现了公钥加密认证体制，Oakley 提供了 IPSec 对等体间达成相同加密密钥的基本模式机制。

SA(SecurityAssocation,安全联盟)是 IKE 用于协商实体通信建立的一种协约。它协定了 IPSec 协议、密钥、转码方式、密钥有效期等信息。IPSec 通信时会构建一个用来维护 IPSec 协议保障数据安全的 SA 数据库（SADB）。SA 信息是单向的，它存储了两设备进行 ESP 通信的会话状态。每台设备同时需要 SA-IN 和 SA-OUT 两组信息，SA-IN 用作处理接受的数据包，SA-OUT 用作处理发出的数据包，SA-IN 和对等方的 SA-OUT 使用相同的加密参数和会话密钥。同一机器的不同 SA 会区分协议，若 AH 和 ESP 同时生效，AH 和 ESP 会产生不同的 SA。IKE（ISAKMP）SA 用于协商 IKE 数据流加密及对等体认证的算法（对密钥加密和 peer 认证），同一个机器最多只能有一个 IKE SA；而 IPSec SA 用于协商对等体之间的 IP 数据流进行的加密算法，同一个机器可以有多个 IPSec SA。

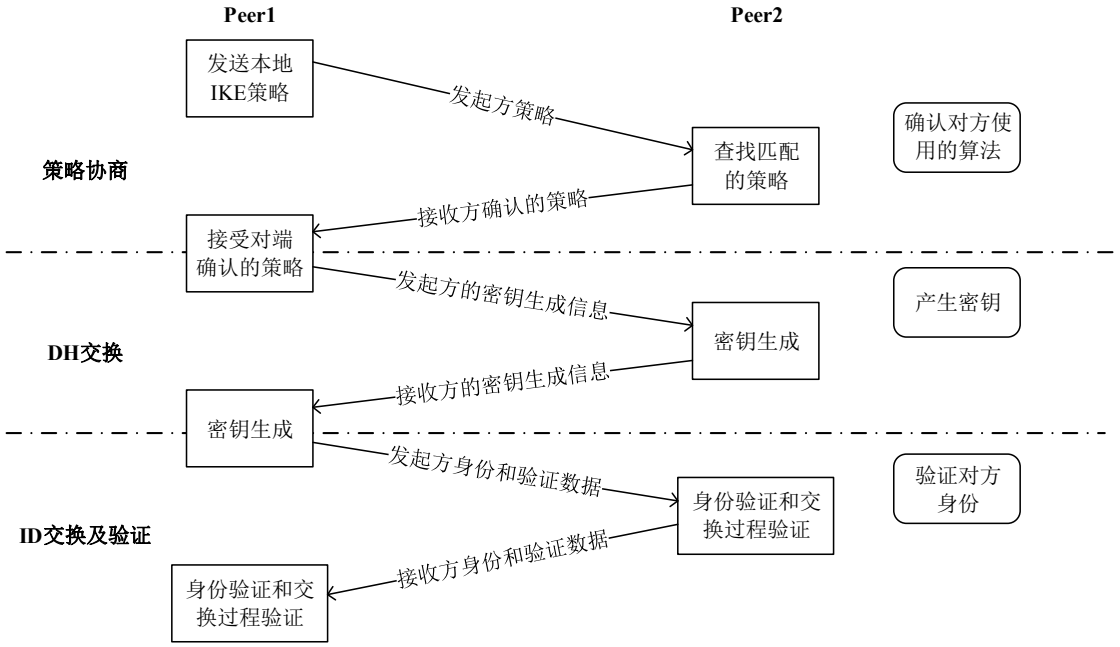

图 2-2 IKE 主模式

IKE 密钥交换模式分为主模式和野蛮模式两种模式。主模式如图 2-2 所示。野蛮模式如图 2-3 所示。野蛮模式为主模式的优化，野蛮模式数据包协商次数更少，速度更快。

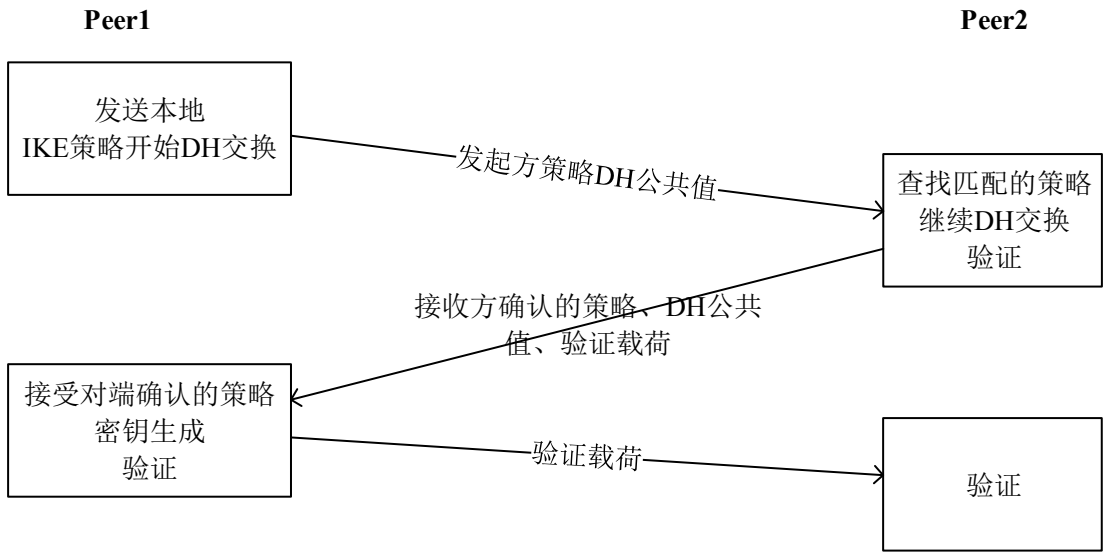

图 2-3 IKE 野蛮模式

IKE 在建立点对点的协商过程有两个阶段：阶段一和阶段二。阶段一用于提供阶段二协商的安全。由于两设备间建立安全的传输连接需要协商使用的加密算法、密钥、封装技术，阶段一首先需要建立安全的管理连接，用作保护加密第二阶段协商过程。阶段二协商在第一阶段加密保护下进行，然后建立安全连接的参数，两设备间形成安全连接，最后两台设备就建立了点对点连接，生成了 SA 信息，可以使用该安全连接来传输数据。SA 用于存储后续传输数据的加密等信息。

IKE 阶段一首先进行 ISAKMP SA 协商，流程如下：

(1)协商对等体间认证的方式（共享密钥或数字证书）

(2)协商加密使用的算法（DES 或 3DES 等）

(3)协商认证使用的算法（MD5 或 SHA）

(4)协商 Diffie-Hellman 密钥组

(5)协商协商模式（主模式或野蛮模式）

(6)协商 SA 生存期

IKE 阶段二用于进行 IPSec SA 协商，流程如下：

(1)协商双方封装技术（ESP 或 AH）

(2)协商加密算法

(3)协商 HMAC 方式（MD5 或 SHA）

(4)协商传输模式（传输模式或隧道模式）

(5)协商 SA 生存期

2.2 XFRM 框架的简介

XFRM 是 Linux 引入的一种基于策略的高扩展性网络安全架构。在 Linux2.6 内核中包含了 PF_KEY，2.4 内核需要打补丁实现。根据 RFC2367 的定义，内核 PF_KEY 实现了安全联盟(SA)和安全策略(SP)的数据库以及用户空间接口[15]。不同系统的 SA 和 SP 实现管理不同，在 Linux 内核中则通过 xfrm 库来实现[16]。

XFRM 框架会捕获到应用程序发往内核的 SA 信息和 SD 信息。SA 用来存安全信息，SA 数据库中包含的信息有 SPI 索引值，目的端 IP，AH 或 ESP，AH 验证算法，AH 验证加密密钥，ESP 验证算法，ESP 验证加密密钥，ESP 加密算法，ESP 加密密钥，隧道或传输模式。SPI 索引值是双方用于索引数据库，手动指定或随机生成的一个值。SP 用来存放 IPSec 哪些流量需要走 IPSec 的规则表，表中包含目的 IP，源 IP，执行协议(AH 或 ESP 或 AH 和 ESP 并存)，源端口，目的端口，工作模式(传输模式或隧道模式)。当主机有数据通过 VPN 发出的时候，数据包会根据 SPD 规则进行匹配，只有匹配到，数据包才会通过 AH 或 ESP 处理。

2.3 DPDK 高性能报文收发平台介绍

2.3.1 DPDK 简介

DPDK(Data planedevelopment kit)是一个用于处理和加速网络数据包的软件库。相比于传统的 Linux 操作系统协议栈，DPDK 通过五个方面来提高数据包处理性能，分别为轮询模式处理数据包，用户态驱动，独占与亲和性，降低访问存开销和软件调优[17]。

轮询模式处理数据包：poll-mode 网卡驱动轮训检查是否有数据到达，避免了因为中断导致上下文切换的开销，提升了收发报文的效率。虽然该方式会导致 CPU 一直处于满负荷运行，但却很适合处理不间断大量数据包。当然在特定情况下仍然支持中断。DPDK 无锁队列是通过内核 kfifo 无锁队列完成的。数据使用生产者消费者模型，一对一，一对多，多对多实现入队和出队，不仅保证了数据的同步，也提高了性能。

用户态驱动：传统协议栈在内核态实现，用户态和内核态交换数据需要内存拷贝和系统调用，不必要的消耗很大。使用 DPDK 可以实现用户态驱动，避免了内存拷贝和系统调用。但用户需自行实现协议栈对数据进行处理和解析。

独占与亲和性：为避免不同核间线程的频繁切换，可以指定某个核心的特定任务，保证 cache 的更高命中率

降低访问存开销：传统协议栈未充分利用 cache 和内存，使用 Hugepage 可以降低 TLB miss，使用内存多通道交错访问提高内存访问有效带宽。大页是 DPDK 通过用户配置，提前在内存预留一块区域，程序可以通过 API 来申请释放大页，大页最小单位可选 2M 和 1G，并且可选择大页的数量。使用 NUMA 来访问尽可能靠近 CPU 的节点内存，跨 NUMA 节点速度相对较慢。

软件调优：数据预取利用了程序的时间和空间局部性原理，软件预先取，cache 行对齐及和 burst 批量处理多元数据，可以一次将 8 个、16 个甚至 32 个报文一次处理，这样可以降低访存次数，提高收发包效率。内存池是 DPDK 巧妙的使用大页技术在用户空间完成的。控制层将数据包扔入内存池，通过指针的方式将控制权转交给下个处理，避免了内存拷贝。

2.3.2 DPDK 加密加速技术

随着网络带宽的增加，数据的实时加密成为新瓶颈，DPDK 提供了多种加密驱动和加密方案[18]。AES-NI:AES-NI 是 Intel 和 AMD 微处理器上 x86 架构的扩展，从硬件上提高了 AES 加密速度，目前 PC 端和服务器的 CPU 对 AES-NI 支持普及率很高。DPDK 支持 aesni-gcm 和 aesni-mb 两种方案。Intel 官方公布的数据：开启 AES-NI 性能比纯软件加密速度提高了 5-8 倍。armv8:在移动端没有 AES 指令时，谷歌推广使用 chacha20 加密算法，此算法在同等配置手机中是 AES 加密算法速度的 4 倍，随后 ARM 在 ARMv8 处理器之后加入了 AES 指令，AES 加密性能反超 chacha20。DPDK 支持多种硬件加密：AMD 加密协处理器 ccp、飞思卡尔的硬件加密（caam-jr、dpaa2-sec、dpaa-sec）、mvsam、octeontx、以及英特尔的 qat 加密加速卡。openssl 加密库。除此之外还支持 kasumi、snow3g、virtio、zuc 等。

2.3.3 用户态协议栈的不足和 DPDK 的应对方案

相比于传统的 Linux 内核协议栈，用户态协议栈优势非常大。数据包到达网卡后会不断产生硬中断，而系统调用和软中断优先级都比硬中断低，产生的消耗比较大。数据包从网卡传送到应用程序需要经历漫长的过程：数据首先要通过 DMA 方式从网卡传到系统内核指定的缓冲区，之后又要将缓冲区接收到的数据拷贝到用户态空间，该过程时间消耗占到 Linux 内核处理数据包整个流程的一多半以上。若将数据和控制分离，用户空间只用于管理内存、处理数据包、调度 CPU，控制指令交由内核管理，就可以解决用户态和内核态频繁切换的问题。除了频繁软硬

中断抢占系统调用产生巨量上下文开销之外，多线程间调度和加锁同样会产生大量上下文切换，DPDK 中采用线程与核心对应来减少线程切换，使用无锁队列避免资源竞争。内存页大小为 4K。操作系统可以为了提高访存速度，避免页失效，但若增加了缓存数量，查询速度又会随之降低，可以使用大页减少映射数目来防止跨内存访问。

2.4 其他协议栈加速技术

2.4.1 零拷贝技术

零拷贝是指的是 CPU 不参与将数据从一个存储区复制到另一个区的任务的计算操作[19]。操作系统的零拷贝（例如设备驱动程序，文件系统和网络协议栈）极大地提高了许多应用程序的性能，因为使用能够进行复杂操作的 CPU 复制数据是一种资源浪费。零拷贝还减少了从用户空间到内核空间的上下文切换次数。 Linux 之类的操作系统支持通过 sendfile，sendfile64 等特定 API 对文件进行零拷贝。创建零拷贝软件的技术包括使用基于 DMA 的复制和通过 MMU 的存储器映射。这些功能需要特定的硬件支持，通常涉及特定的内存对齐要求。零拷贝协议对于高速网络尤其重要，因为内存拷贝会消耗大量 cpu。尽管如此，零拷贝还是有一些初始开销，因此避免编程 IO（PIO）只对大型消息有效果。

在许多对性能要求较高的场景中，复制数据的次数会对的速度产生瓶颈。因此零拷贝算法长期以来一直备受关注。网络通常对性能敏感，并且受到数据复制的影响，因此人们对网络中零拷贝算法产生了兴趣[20]。当进程传输数据缓冲区时，内核必须计算数据的校验和，利用 TCP 分段卸载等技术，能够从数据缓冲区生成数据包。因此可以在硬件级别上可以减少零拷贝操作的时间。在软件方面可以使用 sendfile()系统调用传输文件的内容，而无需通过用户空间复制它们。这种方式在传输页面缓存中的静态数据时效果很好，但它不能用于传输不直接来自文件的数据。如果通常情况下，要传输的数据是某种计算的结果不能使用 sendfile()，并且零复制操作是无法使用。但是，使用它要求用户空间有一些变化。请求零拷贝操作分为两步。一旦建立了套接字，该进程必须调用 setsockopt()设置新的 SOCK_ZEROCOPY 选项。然后可以通过以下调用进行零拷贝传输。数据包接收从内核开始，分配一系列缓冲区，以便在数据包从网络接口出来时保存数据包。通常情况下，内核不知道接下来会从界面显示什么，因此它无法预先知道下一个数据包的预期接收者到达给定缓冲区的人是谁。因此，零拷贝接收的实现必须在分组进入并与开放套接字相关联之后将这些分组缓冲区映射到用户空间存储器中。

反过来，这意味着必须满足一系列约束。将内存映射到进程的地址空间是按页面粒度完成的;没有办法映射页面的一小部分。因此入站网络数据在接收缓冲区中最终必须是页面对齐和页面大小，否则无法将其映射到用户空间。对齐从接口发出的数据包以协议头开始，而不是接收进程感兴趣的数据。实现这种一致性是可能的，但它需要来自网络接口的合作;有必要使用能够在分组进入时将分组报头分成不同缓冲区的网络接口。还需要确保数据以系统页面大小的倍数的块的形式到达，否则将导致部分数据页面。这可以通过在接口上正确设置最大传输单元(MTU)大小来完成。反过来，这可能需要知道输入数据包的确切内容;在测试程序中，Dumazet 将 MTU 设置为 61,512。结果是 15 个 4096 字节的数据页面的空间，加上 IPv6 标题的 40 个字节和 TCP 标题的 32 个字节。Dumazet 的核心是 TCP 套接字的 mmap()实现。通常，在普通文件以外的其他内容上使用 mmap()会创建一系列地址空间，可用于与设备通信等目的[21]。但是，当在 TCP 套接字上调用它时，行为有点不同。如果满足条件（下一个传入数据块页面大小和页面对齐），则包含该数据的缓冲区将映射到调用进程的地址空间，可以直接访问该地址空间。此操作还具有消耗传入数据的效果，就像使用 recvmsg()获取它一样。处理传入数据后，进程应调用 munmap()释放页面并释放缓冲区以获取另一个传入数据包。如果执行失败（例如，只有部分数据可用，或者数据页面没有对齐），则 mmap()调用将失败，返回 EINVAL。如果管道中有紧急数据，也会发生这种情况。通常在这种情况下，调用不会消耗数据，应用程序执行 recvmsg()来获取错误信息。内存映射技巧的零拷贝方案将不如简单复制数据的实现。设置和拆除这些映射涉及相当多的开销。

2.4.2 协议栈硬件加速方案

TOE（TCP 卸载引擎）是 TCP 卸载引擎，也称为 TCP 卸载引擎。它旨在使用网卡上的专用处理器将 TCP 数据包请求传输和处理到主 CPU[22]。TCP/IP 协议目前是互联网数据传输的标准，用于通过局域网（LAN），广域网（WAN）和 Internet 传输信息。TCP/IP 过程可以概念化为层分层堆栈，每个层构建在它下面的层上，提供额外的功能。与 TOE 最相关的层是 IP 层和 TCP 层。IP 层有两个目的：第一个是通过路由过程在 LAN 和 WAN 之间传输数据包，第二是保持与不同物理网络的同构接口。IP 是无连接协议，意味着每个传输的数据包都被视为一个单独的实体。实际上，每个网络数据包属于某个数据流，每个数据流属于特定的主机应用程序。TCP 层将每个网络分组与适当的数据流相关联，并且上层又将每个数据流与其指定的主机应用程序相关联。

大多数 Internet 协议（包括 FTP 和 HTTP）都使用 TCP 来传输数据。TCP 是

面向连接的协议，这意味着两个主机系统必须先建立会话才能在它们之间传输任何数据[23]。虽然 IP 不提供错误恢复-也就是说，IP 有可能丢失数据包，重复数据包，延迟数据包或不按顺序传送数据包 TCP 确保主机系统按顺序接收所有数据包，而不会重复。

由于大多数 Internet 应用程序需要按顺序和可管理的数量交付可靠数据，因此 TCP 是 WAN 和 LAN 的网络协议栈的关键要素[24]。可靠性：TCP 使用校验和错误检测方案，该方案计算数据包报头上的设置位数以及数据包数据，以确保数据包在传输过程中没有被破坏。校验和计算中包括 TCP 伪头，以验证 IP 源和目标地址。有序数据传输：由于属于单个 TCP 连接的数据包可以通过不同的路由到达目标系统，因此 TCP 包含每字节编号机制。此方案使 TCP 协议能够在将数据包传送到主机应用程序之前，将到达其目的地的数据包按顺序返回到它们的发送顺序。流量控制：TCP 监视源系统可以传输的字节数，而不会使目标系统淹没数据。当源系统发送数据包时，接收系统返回确认。TCP 包含一个滑动窗口机制来控制接收端的拥塞。也就是说，当发送方将数据包发送到接收方时，窗口的大小减小;当发送方从接收方接收确认。TCP 通过允许不同的数据流在传输和接收期间混合来适应来自多个发送器的流。它使用称为 TCP 端口的编号标识各个数据流，该端口将每个流与其在接收端的指定主机应用程序相关联。

2.5 本章小结

本章研究了 IPSec 的 ESP 协议和 AH 协议的特点，数据包构造，以及加密算法。并讨论了 IPSec 密钥交换 IKE 的原理、特点和两种模式。研究了主模式和野蛮模式的优缺点和原理。然后介绍了 IPSec 应用程序和内核 IPSec 通信的 XFRM 框架。最后讨论了传统协议栈的不足，并给出 DPDK 解决传统协议栈的解决方案。讨论 DPDK 在加密加速的解决方案。

第 3 章 VPN 的设计方案

通过前一章的技术研究，我们简单了解了 DPDK 相比传统内核的优势和加速的原理。本章首先讨论了需要使用 DPDK 加速的部分以及如何实现，本文讨论了 IPSec 中 IKE 和 ESP 的特点，我们可以设计出 VPN 需要加速的部分 ESP 程序，如何获取 IPSec 的会话密钥等信息。

3.1 IPSec 网关设计与实现

本程序主要涉及 IPSec 的 IKE 和 ESP 两部分，IKE 用作协商密钥和传输规则，而 ESP 用来传输数据的。IKE 又分为 IKEv1 和 IKEv2 两个版本。由于 IKE 阶段数据包较少，实现非常复杂，因此可以使用其他 IPSec VPN 实现 IKE 过程，ESP 阶段数据量巨大，实现相对容易，需要使用 DPDK 进行加速。ESP 部分通信数据加解密使用对称加密算法，若使用软加密，则加密速度成为瓶颈，可使用硬加密来提升加密速度。

对 StrongSwan 和 Linux 内核 IPSec 进行研究，发现内核只实现了 ESP 部分，StrongSwan 主要完成 IKE 部分，两者通信使用 XFRM 框架。由于 ESP 部分需要加密，而 IKE 部分不需要加密，当数据到达网卡后，需要对数据进行分流，ESP 部分交由 DPDK 进行处理，而 IKE 和其他数据需要传入 Linux 内核协议栈。DPDK 和 Linux 内核态协议栈交换报文有两种模式：KNI 或 TUN/TAP。KNI 比 Linux 现有的 TUN/TAP 接口速度更快，因为和 TUN/TAP 相比，KNI 消除了系统调用和其数据拷贝。本程序采用 KNI 来实现 DPDK 和 Linux 内核态协议栈通信。当数据到达网卡后，DPDK 取到数据，获取数据包 IP 头的协议类型，若是 IPv4 或 IPv6 的 ESP 协议，则交由 DPDK 继续处理，否则交由 Linux 内核进行处理。这样就完成了分流功能。

ESP 部分需要 IKE 协商的 SA 安全联盟和 SP 安全策略。在 Linux 内核 2.6 以后，内核实现了 IPSec 传输部分(ESP 和 AH)和 XFRM 框架。因此 IPSec VPN 有两种方案，第一种方案：IPSec VPN 实现 IKE 部分并使用自己的 ESP／AH 程序来处理数据包。第二种方案：IPSec VPN 实现 IKE 部分，并使用 XFRM 框架将 A 安全联盟和 SP 安全策略传给内核，由内核处理 ESP／AH 数据包。本程序实现类似内核的 ESP 部分。使用 StrongSwan 完成 IKE 部分，配置开启 StrongSwan 的 IKEv1 和 IKEv2，配置 StrongSwan 使用内核 IPSec。使用 NetLink 的 libnl 库监听 IPSec VPN 传给内核的 SA 安全联盟和 SP 安全策略，并将其传给 DPDK 的 ESP 部分程序，而

内核虽然收到了 SA 安全联盟和 SP 安全策略，但是由于第一步 ESP 数据被分流给 DPDK 进行处理，内核 IPSec VPN 收不到 ESP 数据。

虽然 DPDK 高速转发性能非常出色，但是 DPDK 没有协议栈。Linux 内核使用 ARP 协议 IP 和 MAC 映射关系。而 DPDK 协议栈处理 ESP 数据包时，由于没有 ARP 协议，难以生成以太网头源 Mac 和目的 Mac。当数据的到达网卡后，IKE 部分数据通过 KNI 被分流到内核。本程序使用内核的 ARP 协议解析 MAC 地址，并解析发往内核的 IKE 数据包，解析 IKE 数据包的以太网头和 IP 头，并保存以太网中的 MAC 地址和 IP 头存入数组，用作后续 ESP 协议通过 IP 查询 MAC 地址来生成以太网头。程序模块总流程图如图 3-1 所示。

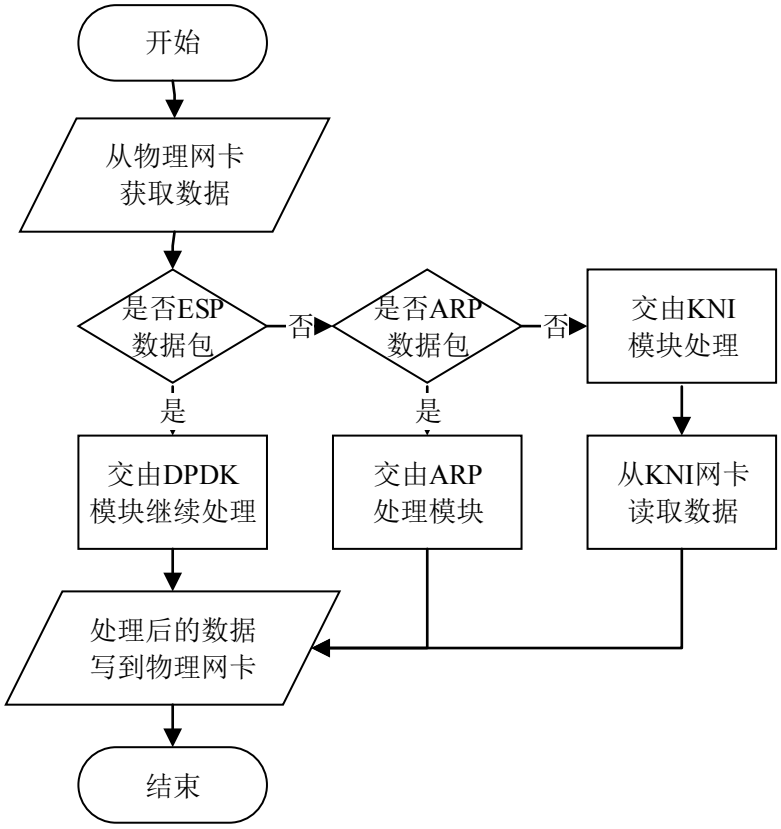

图 3-1 程序模块图

3.2 KNI 内核协议栈转发的设计与实现

如图所示，KNI 在内核注册一个网卡设备，常见的网卡配置工具可以直接配置该网卡信息，通过基于 FIFO 机制来同步控制信息，网络数据使用共享内存来实现。KNI 原理图如图 3-2 所示。

KNI main 函数流程:注册信号中断→配置 EAL 环境→检查参数是否正确→创建内存池 mbuf→初始化 kni 参数→初始化每个 kni 网卡→检查每个 kni 网卡状态→每个核开始运行 main_loop 程序→等待所有核任务结束→释放 kni→释放端口→释放内存

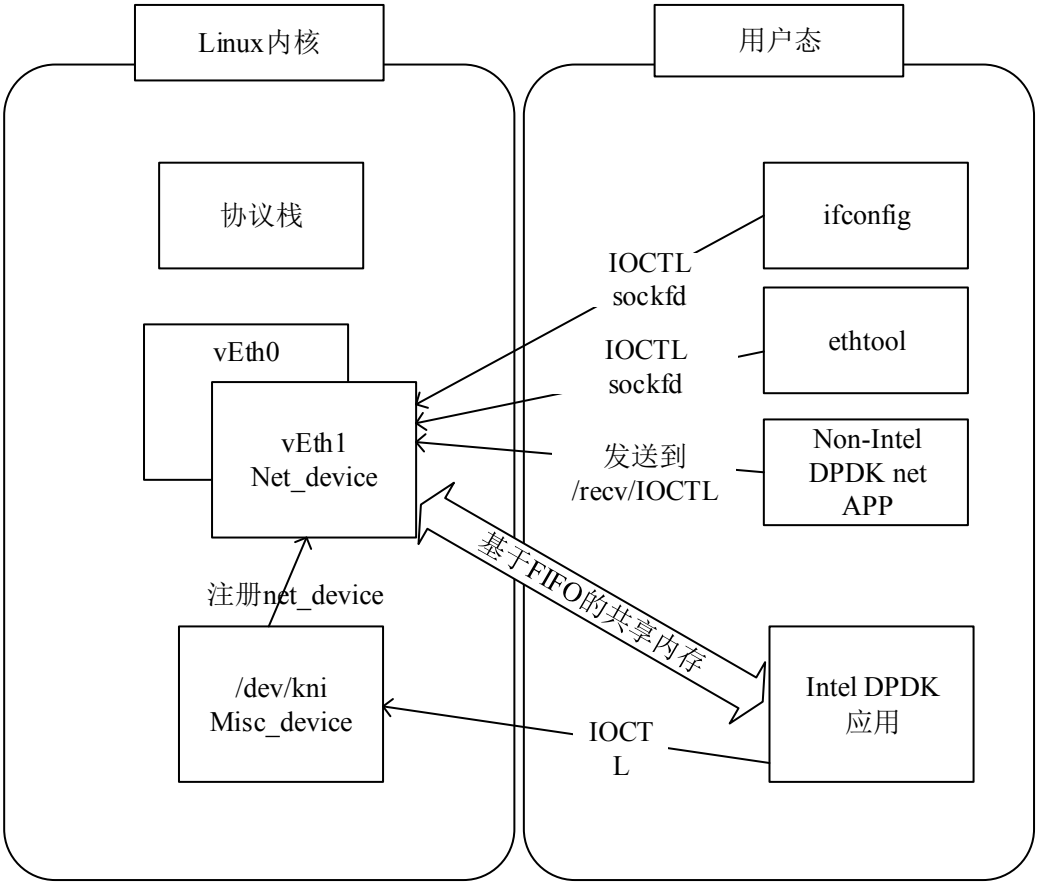

图 3-2 KNI 原理图

3.3 XFRM 监听的设计与实现

DPDK 处理数据包和 XFRM 获取 IPSec 协商结果为两个并行操作，可通过多进程或多线程的方式实现并行操作。若使用多进程，进程间通信方式有：管道，信号，消息队列，信号量，共享内存，原始套接字。

在 Linux 系统中，进程是资源分配的最小单位，线程是调度的最小单位。多进程数据共享复杂，需要 IPC(进程间通信)，但数据同步复杂，而多线程数据在同一个线程中，共享简单，数据同步复杂。和多线程相比，多进程完全复制内存空间，占用空间多，切换复杂，CPU 利用率低，创建销毁速度慢，切换复杂。多进程编

程和调试相对简单。和多线程比，多进程间不会相互影响，而多线程中一个线程挂掉可能导致整个进程挂掉。多进程适合多核心多机分布，扩展到多台机器比较简单，而多线程适合多核分布。结合上述有缺点，本程序只需要两个并行任务，不需要频繁创建销毁，任务间通信数据量不大，因此使用多线程。

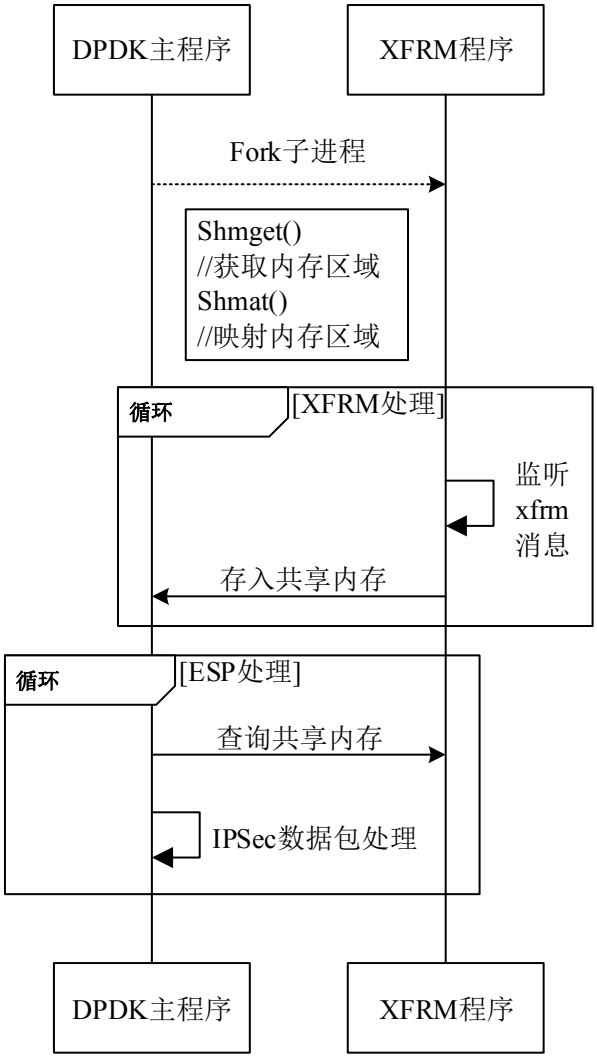

图 3-3 进程间通信图

由于进程间资源隔离，通常进程间不可互相访问。但很多情况下进程通信不可避免，需要进程通过内核或其他进程间通信来完成。常见的进程间通信应用场景有：数据传输、事件通知、共享数据、进程控制、资源共享。管道有三种：无名管道 PIPE、流管道、有名管道 FIFO。无名管道只能父子间通信，并且只能单向流动。流管道可以在父子间双向传输。有名管道可以在多个不相关进程间通信。管道是内核管理的一块缓冲区，空间不大，数据结构为环形以便重复利用。若管

道中没有数据，则读取进程会一直等待，直到写进程写入数据；若管道写满，写进程会一直等待，直到读进程取出数据。当两个进程终止之后，管道也会消失。信号可以发给进程，无需知道进程状态，进程若被挂起，当进程恢复执行时才传给进程。若进程阻塞信号，信号传递讲被延迟，知道取消阻塞才继续被传递。信号是一种异步通信方式，来源有硬件和软件。套接字常用于网络间通信，也可用于进程间通信。由于管道，FIFO，消息队列等都使用内核进行通信，而系统调用是用户空间和内核空间的唯一接口，并且需用户态和内核态进行数据复制，消耗比较大，而本程序对实施性要求比较高，因此这几种方案不可取。而共享内存是通过将同一块内存区映射到不同进程地址空间中，不经过内核，因此共享内存是IPC 中速度最快的，但共享内存需要用户来操作并且同步也需要用户来完成。因此本程序采用最复杂的 mmap 共享内存完成，并使用 CAS 无锁技术来避免加锁，提高性能。进程间通信原理图如图 3-3 所示。

xfrm 使用 netlink 机制来实现 ipsec 用户态程序和内核通信。netlink 是 Linux 内核与用户空间进程通信的一种机制，类似于 UDP 协议，也是网络应用程序与内核通信最常见的接口。netlink 是一种异步通信机制，在内核和用户态之间，传递消息存不用等待存在在 socket 缓冲队列中即可，而系统调用和 ioctl 是同步通信机制。本程序采用 libnl 库来实现捕获应用程序发往内核的 ipsec 的 sa 和 sd。数据流向图如图 3-4 所示。

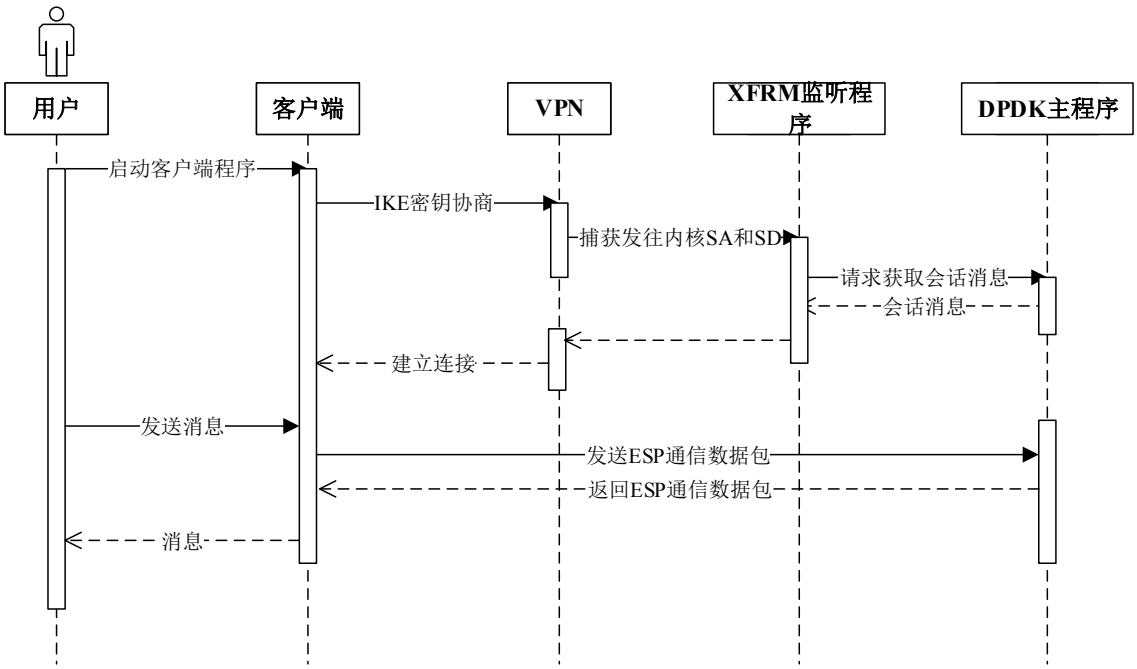

图 3-4 数据流向图

3.4 IP-Mac 映射表的设计与实现

Mac 地址获取如图。当数据不为 ESP 数据包时，其他数据(如 ARP,ICMP,TCP,UDP)走 KNI 网卡，若数据为 UDP 协议，目标地址为 KNI 网卡 IP，端口为 500 时，此数据包为 IKE 通信数据包，可将该数据包的 IP 和端口作为一组 IP 和 MAC 映射表存入数组以备查询使用，KNI 模块流程图如图 3-5 所示。

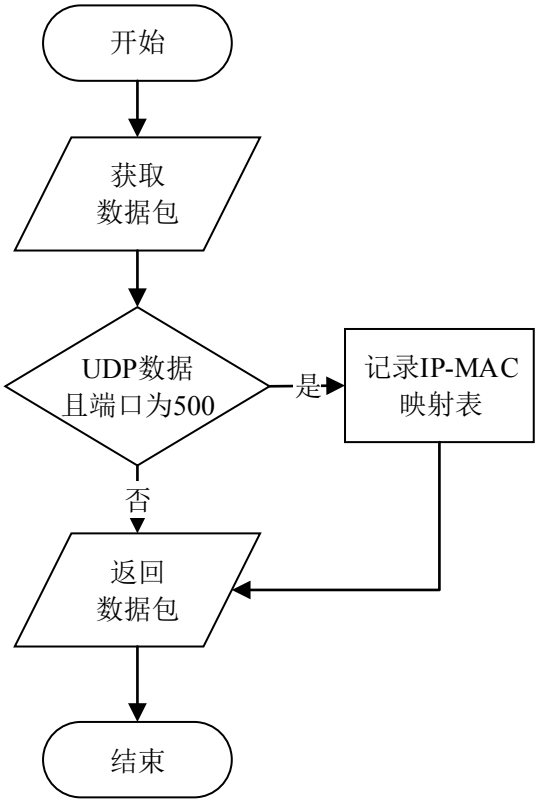

图 3-5 KNI 模块流程图

3.5 本章小结

本章简单讨论了 ESP 数据包和 IKE 数据包，发现 IKE 数据包很复杂，种类繁多，但数据量很少，因此不太需要加速。而 ESP 为数据包简单，数据量大，是加速的重点。内核也是通过应用程序实现 IKE 并将会话密钥通过 XFRM 框架传给内核。然后设计了如何分流 IKE 数据，将数据通过 DPDK 程序传给内核，继而传给 StrongSwan 等程序来完成 IKE，最后讲解了如何捕获应用程序传给内核的会话密钥，以及如何高效实现进程间通信。

第 4 章 VPN 的整体框架

上一章简单介绍了 IPSec VPN 需要使用 DPDK 加速的部分，以及如何将使用 DPDK 加速的 ESP 数据包和 IKE 结合起来。本章介绍了实现用户态协议栈需要完成的工作和难点，程序的整体框架和各框架之间的联系。

VPN 首先要使用网络地址解析模块的 ARP 协议解析局域网主机的 MAC 地址，得到 MAC 地址后，IPSec VPN 客户端使用 IKE 协议进行会话密钥协商，建立连接后使用 ESP 协议进行通信，当数据从 VPN 的局域网转发到广域网时需要使用网络地址转换。同时 IP 数据包需要使用以太网头封装模块封装以太网头。

4.1 地址解析协议模块设计

本程序使用两块网卡，两块 CPU。网卡 1 称作受保护的网卡：IPSec 客户端连接此网卡 IP 进行数据传输，主要用来接收处理客户段发来 IKE 数据和 ESP 数据以及 ARP 数据等，通信为密文数据。网卡 2 称作未受保护的网卡：主要用于处理 ESP 数据包解密后的数据和目的网络通信以及 ARP 数据等，通信为明文数据。

由于 DPDK 程序为用户态协议栈，不能利用内核的协议栈，此处需要和上级网关通信，需要获取上级网关的 MAC 地址，因此需要设计实现 ARP 协议。

DPDK 服务端启动后，首先需要在未受保护的网口向上级网关发送 ARP 请求，以便获取上级网关的 MAC 地址，上级网关受到 ARP 请求后回复 ARP 答复，这样 DPDK 就获得到了上级网关的 MAC 地址。

DPDK 发包时需要封装以太网头，获取上级网关 MAC 地址后，若下次需要向上级网关发送信息，需要封装以太网帧时，首先将自己的 MAC 地址封装到原 MAC 地址，将上级网关的 MAC 地址封装到目的 MAC 地址，将协议类型封装到以太网帧的协议类型。DPDK 服务端还要回复上级网关发给自己的 ARP 请求，因为上级网关需要通过每隔一段时间向 DPDK 服务端发送一个 ARP 请求来确保该主机存活，并保证 DPDK 服务端的 MAC 地址是最新的。ARP 数据包流向图如图 4-1 所示。

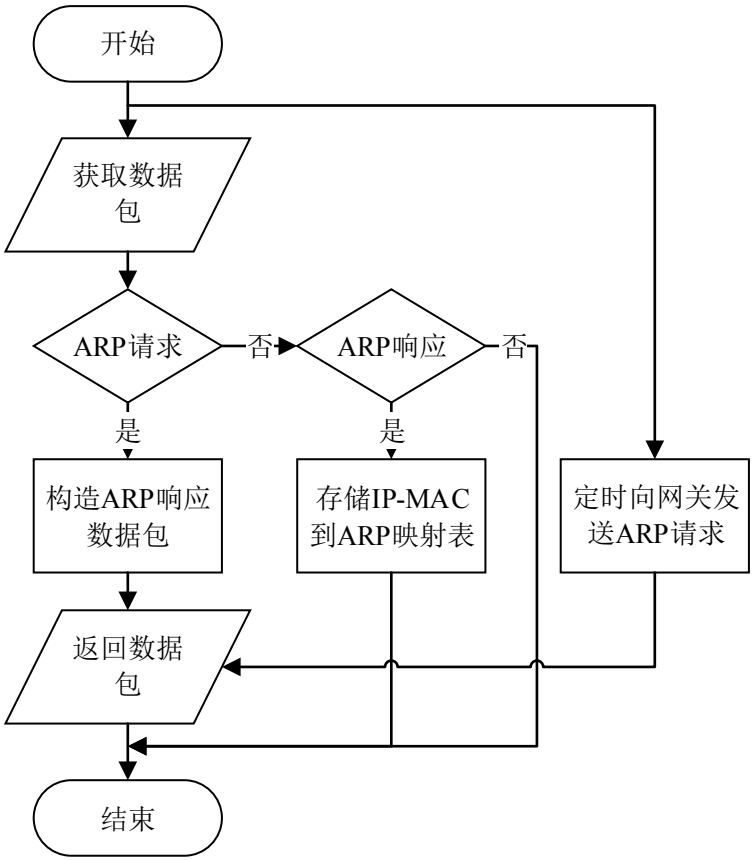

图 4-1 ARP 数据包流向图

4.2 以太网头封装模块设计

CPU1 称作加密核：用来接收受保护网卡的 ESP 数据，对 ESP 密文数据进行解密和完整性校验后，将数据从未受保护的网卡发出去。CPU2 称作解密核：用来接收未受保护网卡的明文数据(UDP 和 TCP 数据包)，将数据进行加密并生成校验信息，进行 ESP 封装后发送给受保护的端口。

IPSec 客户端需要获取目的主机的 MAC 地址，因为 DPDK 程序 IKE 部分通过 KNI 网卡转交给内核。KNI 网卡除了转交 IKE 数据包之外，还可以处理其他数据。当数据到达受保护的网卡后，网卡先判断以太网帧的协议类型，是否为 IP 类型，若非 IP 类型则直接转交给内核，因此此处 IPSec 客户端的 ARP 数据直接交由内核处理。若以太网帧数据类型为 IP，IP 头中协议类型为 ESP，则 DPDK 程序继续处理，否则交由 KNI 网卡转交给内核。DPDK 程序接收到 ESP 数据包时暂时不判断以太网头的原地址和目的地址，发送回复客户端时以太网头的原地址和目的地址必须要正确。因此当收到目的端口为 500 的 UDP 包时，为 IPSec 客户端发送来的 IKE 数据包，因为需要先要通过 IKE 建立连接，才能通信传输 ESP 包，因此可以

将 IKE 数据包以太网帧的原 MAC 地址和原 IP 地址记录存储起来，已备回复 ESP 数据包时封装以太网头。

4.3 网络地址转换模块设计

当 ESP 数据包解密解封装后，需进行 SNAT，将数据包的原 IP 地址的局域网地址转换为广域网地址，原端口也需要随之改变。进行 SNAT 时，首先查询是否存在该条流，若不存在则说明此条流为一个新连接，需要在 NAT 表中插入该流，并为其分配一个广域网端口作为 SNAT 转换后的目的端口。转换后需要对 IP 头和 UDP 或 TCP 头重新计算校验和，然后通过未受保护的端口发出去。当未受保护的端口收到数据时，查询是否存在该条流，若不存在则直接丢弃，若存在需进行 DNAT 后，再进行 ESP 封装加密。此处仅支持 UDP 和 TCP 协议的 NAT，暂不对 ICMP 进行 NAT 处理。NAT 所处位置如图 4-2 所示。

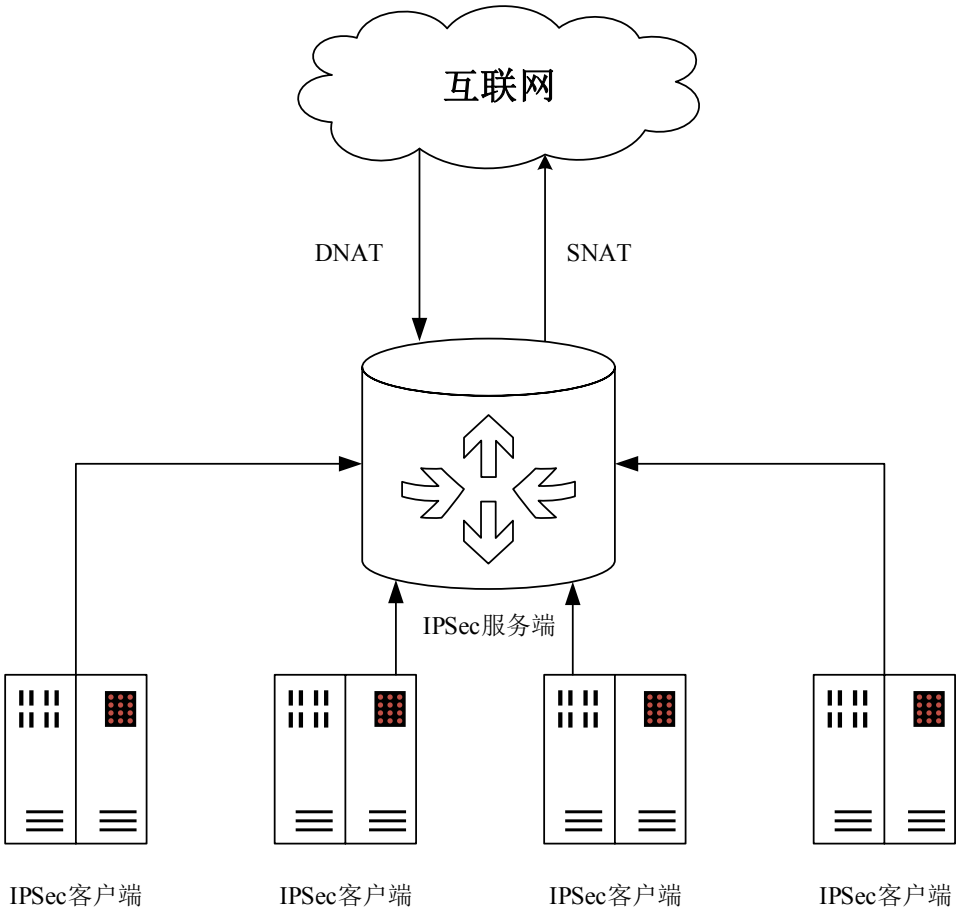

图 4-2 NAT 所处位置

4.4 IKE 数据包的处理模块设计

IPSec 客户端向 DPDK 服务端发送 IKE 请求，被 KNI 网卡转交到内核后，由 StrongSwan 进行 IKE 处理，然后将协商的会话密钥通过 PF_KEY 协议传给内核，本程序通过 libnl 库的 XFRM 框架接受该会话密钥，通过进程间共享内存的方式传给 DPDK 程序。

4.5 本章小结

本章讨论了 IPSec VPN 的整体框架和每个部分所解决的问题。讨论了 IPSec 客户端可以利用 KNI 网卡，使用内核的 ARP 协议。当 IPSec 客户端建立 IKE 时，可将当数据需包中以太头和 IP 头中源 MAC 和源 IP 做为一对存储起来，当 ESP 数据要返回时，可以通过查询该 IP 和 MAC 映射表来获取目的 MAC 地址。简要讲解了 NAT 的作用和原理。当客户端数据经过拆包后，经过 NAT 处理，需要转发给上级网关，因此需要设计 ARP 协议来获取上级网关的 MAC 地址，同时刷新上级网关的 ARP 表。

第 5 章 报文处理流程和设计实现

前两章介绍了 IPSec VPN 的难点和解决方案，并研究了 IPSec VPN 的整体框架。本章详细讲解数据包处理的所有细节。本章研究 ESP 数据包隧道模式和传输模式的区别，数据包中每个字段的作用。实现了 SA 数据库和 SP 数据库。讲解了 NAT 网关所解决的问题和具体实现方式。最后研究了 ARP 数据包构造和通信流程。

5.1 ESP 数据包处理的详细实现

ESP 数据包由安全参数索引（SPI）、序列号（Seq Number），载荷数据、填充、填充长度、下个头、鉴别数据组成。ESP 数据包构造如图 5-1 所示。

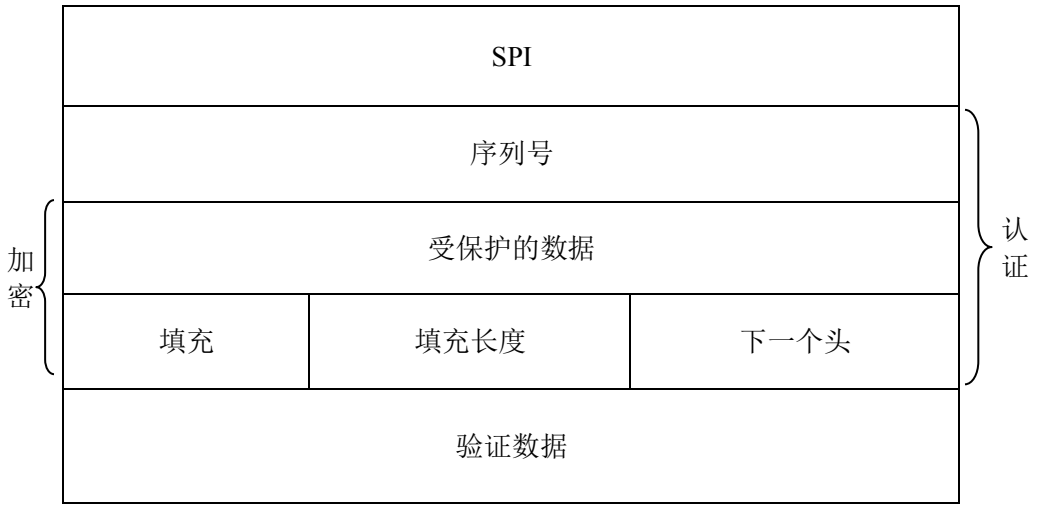

图 5-1 ESP 数据包构造

SPI 为 32 位，是一个用来确定唯一标示的安全联盟。

Seq Number 为 32 位，用来避免接受方的重放攻击。

载荷数据为加密后的数据包，隧道模式会连同 IP 头一起加密。

填充（Padding）长度为 0-255 个字节，隐藏载荷数据长度并将明文扩充到加密长度

填充长度的长度为 8 位

下一个头部标志下一个被加密数据的头部的类型

鉴别数据在 ESP 数据包前面的基础上计算出完整性校验值。

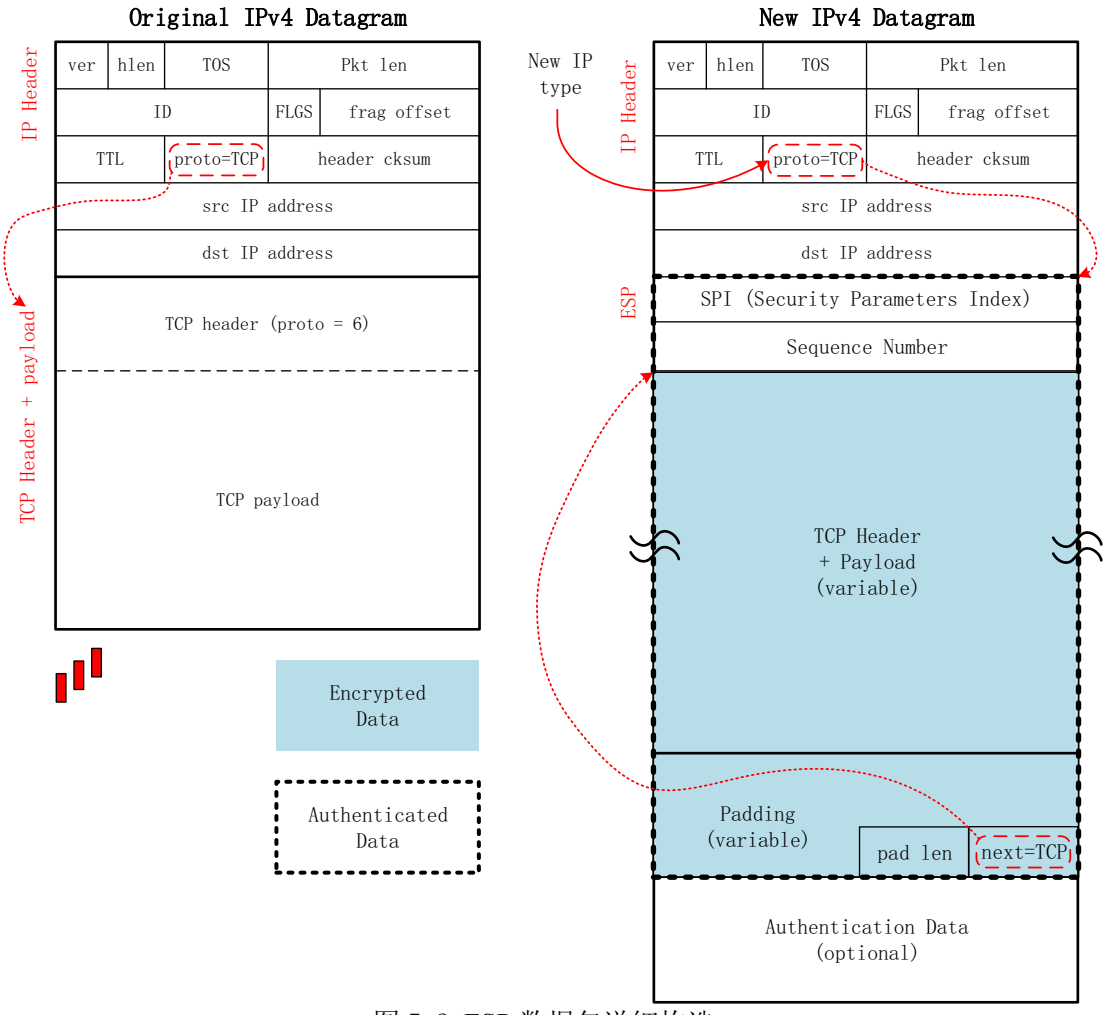

图 5-2 ESP 数据包详细构造

ESP 数据包封装前后区别如图 5-2 所示。

ESP 封包处理:

(1)如果是隧道模式，跳过这一步，如果是传输模式，获取 IP 头部长度和协议类型，以备构造新 IP 使用。

(2)如果是隧道模式，载荷长度为数据包的长度加上 2，因为数据包的 IP 头信息会被放到 ESP 头部中，原数据包被封装到 ESP 包中，头部会被丢弃掉。如果是传输模式，载荷长度为数据包的长度减去 IP 头长度加上 2。填充长度的长度为 1 和下个头的长度为 1，都算在载荷长度中，因此要加 2。然后将基于这个长度，将数据扩展到块长度，为最终的载荷长度。填充长度为载荷长度加上 IP 头长度减去数据包原来的长度。也就是最后数据包填充的长度。

(3)如果数据包总长度（IP 头长度+ESP 头长度+载荷长度+鉴别）超过数据包最大大小 MTU，则数据包要么重组，要么丢弃。本程序中数据包超过 MTU 选择

丢弃。

(4)将数据包扩展填充长度和认证长度，这部分数据为载荷数据，也就是需要加密的部分，接下来对载荷层数据包进行封装。

(5)将原数据包的 ttl 改为-1，获取原数据包的服务类型(TOS)作为新 IP 头的 TOS。TOS 是用来设置 IP 包的优先级和 QoS 选项。构造新的 IP 头。并将协议类型置为 ESP 类型。

(6)将 SPI 加入 ESP 头，将 SEQ 自加 1，加入 ESP 头

(7)加密内容包含载荷数据(传输模式为去掉 IP 头的数据，隧道模式为原数据)，填充数据和填充长度，下个头，认证信息，其中数据的倒数第二个字节为填充长度，第一个字节为下个头，填充内容为从 1 到 255 的字节码。

(8)IV 为 AES 加密的初始向量，是一个固定长度的输入值，为区块大小，要求他是随机或伪随机数，加密的时候会产生 IV 位。第一个 IV 由用户提供，其他 IV 都是自动生成的。IV 放在 ESP 的 SPI 和 Seq Number 之后，IV 长度为 16，是未加密的，但当做密文的一部分

(9)认证内容从 IP 头之后开始，到认证尾部(认证结果位置之前)，包含 ESP 头部和明文载荷数据，明文扩展数据，明文扩展长度，明文下个头，然后选择认证算法，对数据进行认证，并放在数据包最尾部，最后进行加密

ESP 解包处理:

(1)首先计算 IP 头部长度，载荷数据长度为数据包总长度减去 IP 头长度再减去 IV 长度，减去认证长度，使用指定的加密密钥和加密算法对数据进行解密。载荷数据应该为块大小的倍数，并且大于 0，否则载荷数据又问题，需要丢弃。

(2)使用 IV 值，从 ESP 头部之后(SPI 和 Seq Number 之后)对数据进行解密，认证数据从 IP 头部之后开始进行解密

(3)对数据进行认证，认证从 IP 头之后开始，认证的长度包含 ESP 头(SPI 和 Seq Number)，IV 长度，载荷长度。使用密钥解密数据包中的认证信息，并和计算的认证结果进行对比，判断数据包是否校验通过

(4)对解密后的数据进行处理，获取下个头数据，下个头的位置为数据包的长度减去校验数据长度减下个头长度 1。获取填充数据长度，填充数据长度位置为下个头位置减去填充数据长度的长度 1。

(5)填充数据位置为填充数据长度的位置减去填充数据长度，对填充数据进行判断，每个填充数据应为填充位置的字节码（从 0 到 255）。如果填充数据不正确，说明数据包有问题，直接丢弃。移除填充数据，填充数据长度，下个头，认证信息

(6)如果是传输模式，将 ESP 数据包的 IP 头做为解密后新数据包的头部，新数据包的协议类型改为下个头，IP 头长度改为解密后载荷数据的长度

(7)如果是隧道模式，IP 头包含载载荷数据中，因此不用单独处理 IP 头

5.2 安全关联数据库的设计与实现

安全关联数据库的每条信息决定了 SA 的参数，当 IPSec SA 被创建，SADB 更新所有关于 SA 的参数，当一个数据包到达时，SADB 根据外层 IP 头部的目的 IP 地址，SPI 和 IPSec 封装协议（ESP / AH）查询数据库来获取对应的 SA，根据 SA 的参数来确定怎样处理这个 IPSec 数据包。对于外来 IPSec 的数据包，则由 SPD 所关联的 SA 来获取。IPSec 数据包处理流程如图 5-3 所示。

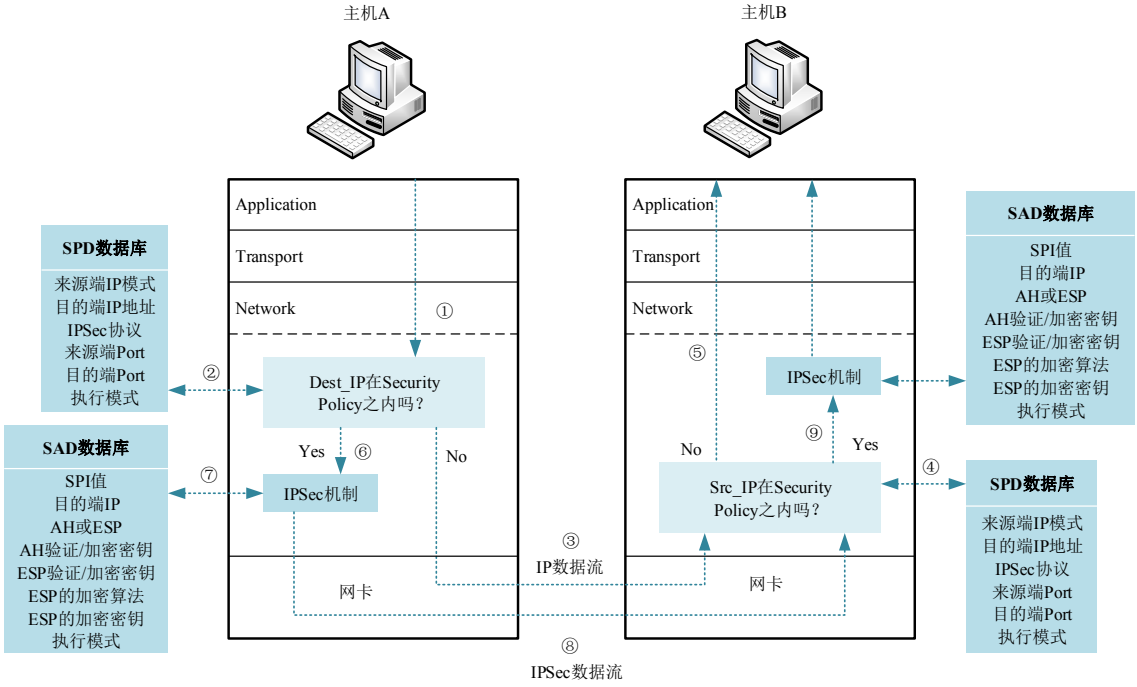

图 5-3 IPSec 数据包处理流程

SA 中包含的信息有：

(1)序列号（32 位）

(2)序列号溢出位（溢出之后重新开始）

(3)防重放窗口（默认 64 位）

(4)SA 生存时间，有软时间或硬时间，SA 到期前一段时间会重新生成新的密钥，从密钥生效到申请密钥这段时间叫软时间

(5)传输模式或隧道模式

(6)ESP 验证算法

(7)ESP 加密算法

(8)MTU

SADB 存储内容如表 5-4 所示.

表 5-4 SADB 存储内容

SPI	目的 IP	执行协议	ESP 验证算法	ESP 验证密钥	ESP 加密算法	ESP 加密密钥	工作模式
0x201	192.168.0.10	ESP	hmac-sha1	rrrrrrrrrr	3des-cbc	kkkkkkkkk	隧道模式
0x301	192.168.0.1	ESP	hmac-sha1	aaaaaaa	3des-cbc	aaaaaaaa	隧道模式
0x401	192.168.0.20	ESP	hmac-md5	sssssssss	3des-cbc	bbbbbbb	隧道模式
0x501	192.168.0.1	ESP	hmac-md5	pppppp	dec-cbc	ccccccccc	隧道模式

5.3 安全策略数据库的设计与实现

选择器会选择指定的流加密，可通过源 IP，目的 IP，名字，传输协议，源端口号和目的端口号进行选择，并有三种处理方式：1.旁路，保证原封不动 2.丢弃 3.使用 IPSec 加密.。SPDB 存储内容如表 5-5 所示。

表 5-5 SPDB 存储内容

来源 IP	目的 IP	执行协议	源端口	目的端口	工作模式
192.168.0.1	192.168.0.10	ESP	任意端口	任意端口	隧道模式
192.168.0.10	192.168.0.1	ESP	任意端口	任意端口	隧道模式
192.168.0.1	192.168.0.20	ESP	80	任意端口	隧道模式
192.168.0.20	192.168.0.1	ESP	任意端口	80	隧道模式

5.4 网络地址转换算法的设计与实现

NAT 概述:随着互联网的发展，用户和设备越来越多，IPv4 地址逐渐枯竭，用户数远大于 IPv4 地址容量。从长远来看，只能更换协议，但更换协议时一个长期的工作，不能一下就完成。短期来看，网络地址转换（NAT）可以较好的解决这一问题。

通常 Linux 下的 VPN 服务端先将数据从网络接收到，然后经过解包后发给虚拟网卡，然后虚拟网卡使用 iptables 实现 NAT 并将数据转发出去，之后数据原路返回，这个流程复杂并且速度慢。

NAT 将很多个私有 IP 地址转换为一个或多个公网地址，实现局域网主机和广域网主机的通信。这样就减少了公网地址的分配，实现了 IP 地址的复用，消减了 IP 地址空间不足，并且 NAT 也可以隐藏局域网内部，极大保护了局域网安全。但 NAT 也存在其不足，NAT 针对从内向外部通信，需要局域网主机先向外发出请求，才可处理外部的回复。外网不能直接和内部通信，如需要和内部主机通信，需要使用端口映射或 DMZ 主机方式，配置较为麻烦。BT 下载或者视频通信时需要点对点通信，两个处于局域网间的机器通信将不方便，此时可通过 P2P 打洞穿透 NAT，但这需要 NAT 支持。NAT 原理图如图 5-6 所示。

Direction	Before NAT	After NAT
Outbound	192.168.1.2:1111	20.1.1.1:1001
Outbound	192.168.1.2:2222	20.1.1.1:1002
Outbound	192.168.1.3:1111	20.1.1.1:1003

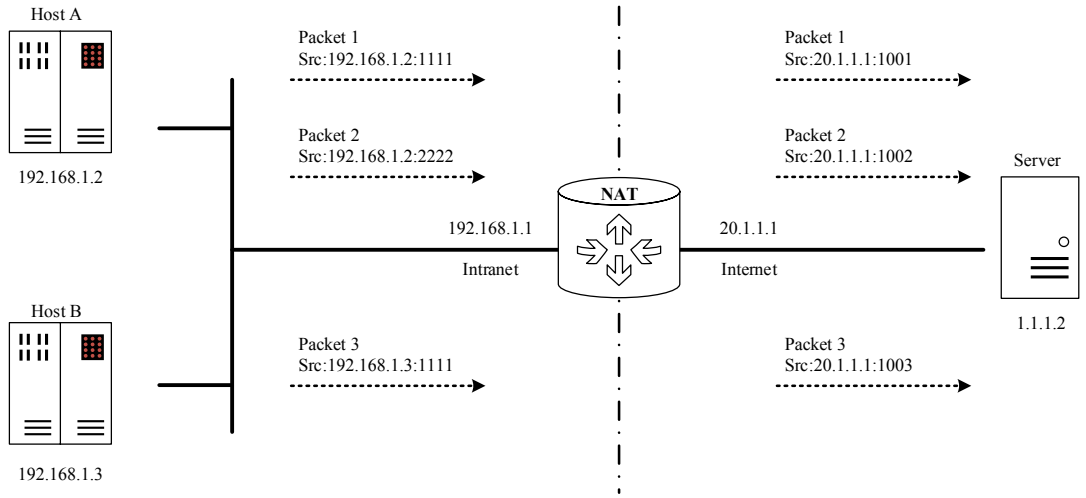

图 5-6 NAT 原理图

NAT 算法:当数据到来时，首先判断数据包的流向，当源 IP 和内网 IP 同在一个网段，目的 IP 和内网 IP 不在同一网段时，数据为从内网发向外网的数据，需要 SNAT。当源 IP 和内网 IP 同在一个网段，目的 IP 和内网 IP 在同一个网段，数据是从内网发向内网的数据，需要 FORWARD。当源 IP 和内网 IP 不在同一个网段，目的 IP 为外网 IP 时，查询 NAT 表中确实存在该条流，数据为外网发向内网的数据，需要做 DNAT 操作。

判断是否处于同一网段算法:判断是否处于同一个网段的方法为将两个要判断

的 IP 和子网掩码同时做与操作，与之后的结果相同，则说明两个 IP 处于同一个网段。

五元组概念:做 NAT 时，需要根据五元组在哈希表中查找，五元组为源 IP，源端口，目的 IP，目的端口，协议类型，可以根据五元组确定唯一一条流，本程序使用 uthash 作为哈希算法。

SNAT 算法:当做 SNAT 操作时，首先根据数据包的五元组在内网五元组中查找。若不存在该条数据，说明该数据为一条新的连接，需要为该条流分配一个外网端口，并将内网五元组和外网五元组作为一组放在一起，以备 DNAT 查询。将数据包的五元组做为内网五元组，而外网五元组的目的 IP 为外网 IP，目的端口需要挑选一个端口，协议类型为数据包的协议类型，源 IP 为数据包的目的 IP，源端口为数据包的目的端口，然后将内网五元组和外网五元组放在一起存到哈希表中。最后将数据包的源 IP 改为外网 IP，将数据包的目的端口改为外网五元组中的目的端口，即分配的那个外网端口。并刷新该数据的存活时间，即将数据包的 TTL 值改为定义的存活时间。

分配外网端口算法:分配外网端口的准则为外网五元组不能重复，否则外网五元组不能找到唯一的内网 IP 和内网端口。从指定的端口范围中随机选取一个，保证生成的五元组和外网五元组中的不重复即可。早期端口号分为三大类，公认端口号从 0 到 1023，用于系统保留使用。注册端口从 1024 到 49151，用于应用程序分配提供公网服务。动态或私有端口从 46152 到 65535，服务不应该分配这些端口，用于操作系统为客户端分配源端口，但现在的操作系统很少遵循该规定，操作系统通常从 1024 起开始分配动态端口。

DNAT 算法:当做 DNAT 时，根据数据包的五元组查询外网五元组的哈希表，。若未查到，则无法找到内网 IP 和端口，会被 NAT 丢弃。通常这种情况说明内网为发送这样一条流，为外网发向内网的一条流。或内网发送的这条流已经超时被丢弃，nat 表中已删除。若 HASH 表中存在，则将数据包的目的 IP 改为内网五元组的源 IP，目的端口改为内网五元组的源端口。

NAT 表超时算法:NAT 的 HASH 表需要删除超时的流，否则 NAT 表会越来越大，分配的外网端口也会被使用枯竭。当每次做 SNAT 后，该条 NAT 的 ttl 值会被改为定义的存活时间。定时器会定时将 NAT 表中的所有 ttl 值做自减，当值减为 0 时，说明该记录已超时，会将该条 NAT 删除。因此不论 TCP 还是 UDP，若要保持长连接，若长时间无通信数据，需要定期发送心跳包。

5.5 地址解析协议算法的设计与实现

ARP 概述:ARP 协议全称为地址解析协议，用于实现从 IP 地址到 MAC 地址的映射，在网络通信中，数据包在广域网中主要需要根据 IP 地址进行传输判断，到达路由器后，根据路由表进行转发。而在局域网中，不仅需要 IP 地址封装，也需要 MAC 地址的封装，交换机或主机路由器等设备根据 ARP 表对数据进行转发，普通模式下，若数据包的目的 MAC 地址不为自己，数据包会被网卡直接丢弃，混杂模式下网卡会收到目的 MAC 不为自己的数据。ARP 协议非常快，实现也很简单，但也最不安全，ARP 请求以广播模式在局域网中发送，局域网任何机器都可以进行回复，也可以更改其他机器的 ARP 表，他没有任何做任何验证处理，因此网络扫描，内网渗透，中间人拦截，流量控制，流量欺骗基本都和 ARP 协议有关。ARP 为根据 IP 地址解析 MAC 地址，RARP 为根据 MAC 地址解析 IP 地址。ARP 数据包构造如图 5-7 所示。

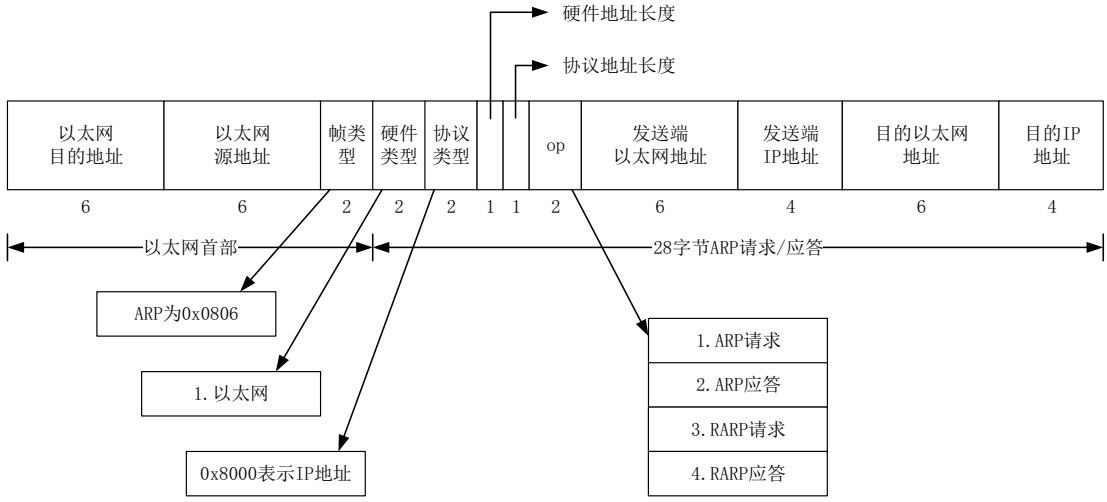

图 5-7 ARP 数据包构造

ARP 数据包结构:以太网头包含 6 个字节的源 MAC 地址，6 个字节目的 MAC 地址，2 个字节的协议类型，协议类型为 ARP 协议。ARP 数据包包含 2 个字节的硬件格式，2 个字节的协议地址格式，1 个字节的硬件地址长度，1 个字节的协议地址长度，2 个字节的请求类型（ARP 请求或 ARP 响应），6 个字节的发送方 MAC，4 个字节的发送方 IP，6 个字节的接收方 MAC，4 个字节的接收方 IP。数据包格式如图 5-8 所示。

硬件类型		协议类型
硬件地址长度	协议长度	操作类型
发送端 MAC 地址		发送端 IP 地址
目的端 MAC 地址		目的端 IP 地址

图 5-8 ARP 数据包格式

发送 ARP 请求：通常 IPv4 的 ARP 数据包的硬件格式，协议地址格式，硬件地址长度，协议地址长度通常为固定值，若需要查询对方 MAC 地址，以太网头源地址为本机网卡地址，目的地址为广播地址，ARP 数据包中发送方 IP 和 MAC 地址为本机网卡中信息，目的 IP 为要查询的 IP，MAC 地址置为 0。

处理 ARP 响应：当发出 ARP 请求后，目标主机收到会给以响应，响应的数据包以太网头的目的 MAC 为本机，源 MAC 为回复的机器，若接收方的 MAC 地址和 IP 地址为自己，ARP 类型为 ARP 响应，并且发送方为想要查询的 IP，那么发送方的 MAC 地址则为查询的 MAC。当本机发出 ARP 请求时，目标机会将本机的信息添加到自己的 ARP 映射表中，然后再予以回复。

处理 ARP 请求：对方每发送几个数据包后，都会清空本机器的 ARP 映射，并发送一个 ARP 请求，来确保对方主机存活，接收到的数据包为，以太网帧中发送方 MAC 为本机 MAC，接收方 MAC 为目标机 MAC，ARP 包中请求类型为 ARP 请求，发送方 MAC 和 IP 为目标机信息，接收方 MAC 为空，IP 为本机 IP，我们需要回复的信息为，不修改以太网帧，发送方 IP 和 MAC 为本机信息，接收方 MAC 和 IP 为目标机。对方收到后会重新刷新 ARP 映射表。

5.6 校验和算法的实现

为了保证数据包的完整性，IP 头，TCP 头，UDP 头，ICMP 头等数据包都会通过校验算法生成校验信息。

IP 头格式如表 5-9 所示。IP 头校验信息生成算法：

(1)用零将两个字节的校验位置填充

(2)校验的长度为 20 位 IP 头，每次取两个字节(16 位)的数据进行不断累加，累加的结果存在 32 位的数中

(3).将累加的结果的高 16 位和低 16 位进行相加

(4)将最后相加的结果取反做为校验值

<table>
<tr><td>版本
(4 位)</td><td>头长度
(4 位)</td><td>服务类型
(8 位)</td><td colspan="2">封包总长度(16 位)</td></tr>
<tr><td colspan="3">封包标识(16 位)</td><td>标志
(3 位)</td><td>片断偏移地址(13 位)</td></tr>
<tr><td>存活时间(8 位)</td><td colspan="2">协议(8 位)</td><td colspan="2">校验和(16 位)</td></tr>
<tr><td colspan="5">原 IP 地址(32 位)</td></tr>
<tr><td colspan="5">目的 IP 地址(32 位)</td></tr>
<tr><td colspan="3">选项(可选)</td><td colspan="2">填充(可选)</td></tr>
<tr><td colspan="5">数据</td></tr>
</table>

图 5-9 IP 头格式

UDP 校验和计算数据包如图 5-10 所示。UDP 或 TCP 校验和计算方法:

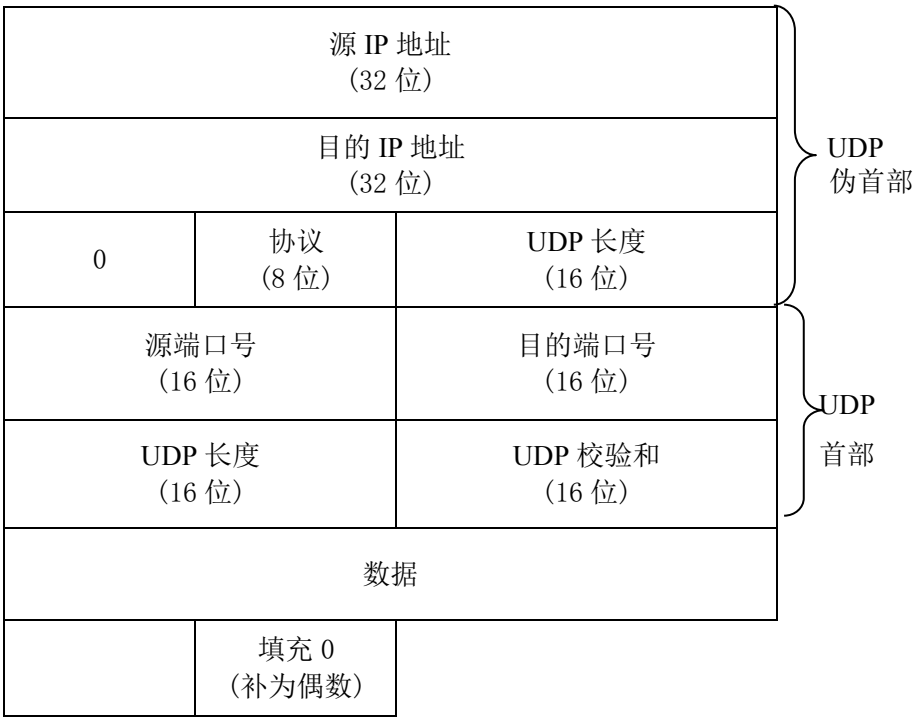

图 5-10 UDP 校验和计算数据包

(1)用零将两个字节的校验位置填充，若数据包长度为奇数，末尾填充一个零

(2)生成伪 IP 头，伪 IP 头包含源 IP，目的 IP，UDP 数据长度，一个零字节做为保留位，协议类型

(3)将伪 IP 头，UDP 头，UDP 数据放在一起计算校验验和

(4)每次取两个字节(16 位)的数据不断累加，累加的结果存在 32 位的数中

(5)将累加的结果的高 16 位和低 16 位进行相加

(6)将最后相加的结果取反做为校验值

5.7 本章小结

上一章简单讲解了 IPSec VPN 中各模块及其解决的问题，本章详细讲解了数据包的构造。当 IPSec VPN 完成 IKE 后，连接正式建立，ESP 数据包经过分流经由 DPDK 程序处理。网关首先根据 SPI 查询 SA 数据库，获取其加密密钥和认证密钥，然后查询 SP 数据库，判断这个数据包需要使用加解密还是直接放过还是丢弃。然后解密数据包，拆包并对数据包进行认证，判断数据包是否被篡改过。最后使用 SNAT 对数据包的源端口和源 IP 进行转换，由于源 IP 和端口改变，重新计算 IP 数据包和 UDP／TCP 数据包的校验和，然后将上级网关的 MAC 地址和本机的 MAC 地址封装到以太网头后，将数据包发出去。经过目的地服务器的处理后回应数据包，回应的数据包首先进行 DNAT 处理，然后重新计算 IP 头和 UCP／TCP 数据包的校验和，最后查询 SP 数据库，根据规则匹配到对应 SA，然后使用 SA 的加密密钥和认证密钥对数据进行封装，封装后的 ESP 数据包，根据其 IP 地址查询 IP 和 MAC 映射表，获取客户端的 MAC 地址，将本机 MAC 和客户端的 MAC 地址封装到 ESP 数据包的以太网头后，将数据发给客户端，实现 ESP 整个流程。

第 6 章 环境配置与测试分析

前面讲解了基于 DPDK 的 IPSec VPN 的实现方案和原理。本章对程序性能进行测试，测试使用 DPDK 后，对 IPSec 处理数据性能究竟有多大提升，并检测 IPSec VPN 服务端的稳定性。

6.1 环境配置

6.1.1 安装 DPDK 所需软件

安装基本编译软件:gcc(version>=4.9),make,cmp,sed,grep,arch
安装依赖 C 库:
libc headers,glibc-devel.i686/libc6-dev-i386,glibc-devel.x86_64/libc6-dev,glibc-devel.ppc64
安装内核源码:
yum install kernel-headers-`uname -r`
yum install kernel-src-`uname-r`
yum install kernel-devel-`uname-r`
安装依赖软件和库:
python(version>=2.7 or version>=3.2)
pciutils，libudev-devel,boost-devel,zlib-devel,openssl-devel,libpcap-devel,lua-static

6.1.2 DPDK 源码编译安装

```
# vi ./config/common_base     #修改编译参数
CONFIG_RTE_LIBRTE_KNI=y  #编译 KNI 网卡驱动
CONFIG_RTE_LIBRTE_PMD_OPENSSL=y  #编译 OPENSSL 库
# export RTE_SDK=/root/ipsec_demo/dpdk-17.02  #添加环境变量
# export RTE_TARGET=x86_64-native-linuxapp-gcc
# make install T=x86_64-native-linuxapp-gcc          #编译
```

6.1.3 配置 strongswan 证书

(1)生成根证书
ipsec pki --gen --outform pem > ca.key.pem

ipsec pki --self --in ca.key.pem --dn "C=CN,O=T2,CN=Test CA" --ca --lifetime 3650 --outform pem > ca.cert.pem > ca.cert.pem

(2)生成服务器证书

IP=192.168.100.1　#配置服务器连接的地址，配置错误无法连接成功

ipsec pki --gen --outform pem > server.key.pem

ipsec pki --pub --in server.key.pem --outform pem > server.pub.pem

ipsec pki --issue --lifetime 1200 --cacert ca.cert.pem --cakey ca.key.pem --in server.pub.pem --dn "C=CN,O=TZ,CN=Test Server" --san="$IP" --flag serverAuth --flag ikeINtermediate --outform pem > server.cert.pem

(3)生成客户端证书

ipsec pki --gen --out form pem > client.key.pem

ipsec pki --pub --in client.key.pen --outform pem > client.pub.pem

ipsec pki --issue --lifetime 1200 --cacert ca.cert.pem --cakey ca.cert.pem --in client.pub.pem --dn "C=CN,O=TZ,CN=Test client" --outform pem > client.cert.pem

(4)配置环境

#加载 kni 驱动

rmmod rte_kni

insmod $RTE_SDK/x86_64-native-linuxapp-gcc/kmod/rte_kni.ko

#加载驱动

modprobe uio

insmod $RTE_SDK/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko

#配置大页

echo 1024 >

/sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages

(5)启动程序

./build/app/ipsec-secgw --master-lcore 0 -l 0,1 -n 1 --socket-mem=2048,0 --vdev "crypto_openssl" -w 02:00.0 -w 03:00.0 -- -k 0x1 -p 0x3 -P -u 0x01 --config="(0,0,0),(1,0,1)" -f ipsec-test.cfg #works

(6)配置 kni 网卡 IP

ifconfig vEth0 192.168.100.1/24
sleep 1
ifconfig vEth0 up
ifconfig vEth0 mtu 1450
ifconfig vEth0 hw ether 00:11:22:33:44:55
ifconfig vEth0
ping -c 10 192.168.100.2

(7)客户端安装 SA 证书

(8)客户端连接 IPSec VPN

6.2 测试环境

设备连接如图，客户端电脑和服务端软路由直连，设备配置参数如表 6-1 和表 6-2 所示。

表 6-1 服务端配置数据

环境类型	要求
CPU	Intel® Celeron® Processor J1900
内存	8 GB 1600MHz DDR3L
网卡	Intel® Ethernet Controller I211-AT
系统	CentOS Linux Minimal 7.3

表 6-2 客户端配置数据

环境类型	要求
CPU	Intel(R) Core(TM) m3-6Y30
内存	8 GB 1867MHz LPDDR3
网卡	绿联 typec 扩展坞(千兆网卡芯片 AX88179)
系统	MacOS Sierra version 10.12.6

6.3 测试对象

测试对象与算法如表 6-3 所示。

表 6-3 测试对象与算法

测试对象	结果
实验组(基于 DPDK 的 IPSec VPN)	Intel(R) Core(TM) m3-6Y30
对照组(基于 Linux 内核的 IPSec VPN)	8 GB 1867MHz LPDDR3
加密算法	绿联 typec 扩展坞(千兆网卡芯片 AX88179)
认证算法	MacOS Sierra version 10.12.6

6.4 测试工具

本文使用 iperf3 对网关进行发包测试其性能。

6.5 测试流程

服务端首先使用生成的证书信息启动 StrongSwan 程序，并设置 StrongSwan 使用 xfrm 框架而非自己的 ESP 协议。然后设置 iptables 放行 UDP 端口为 500 的数据包，ESP 数据包。然后启动本程序，并启动 KNI 网卡，并配置 KNI 网卡的 IP，MTU，MAC 地址等信息。最后使用 IPSec 客户端连接 IPSec 服务端。连接成功后使用 iperf 进行性能测试。客户端接在受保护的端口，服务端接在未受保护的端口，服务端启动 iperf 服务端，客户端启动 iperf 客户端。

使用 UDP 协议，更改 iperf 参数使程序发送不同长度的数据包，然后可测出不同包长下，包转发率的大小。包转发率测试结果如图 6-4 所示。

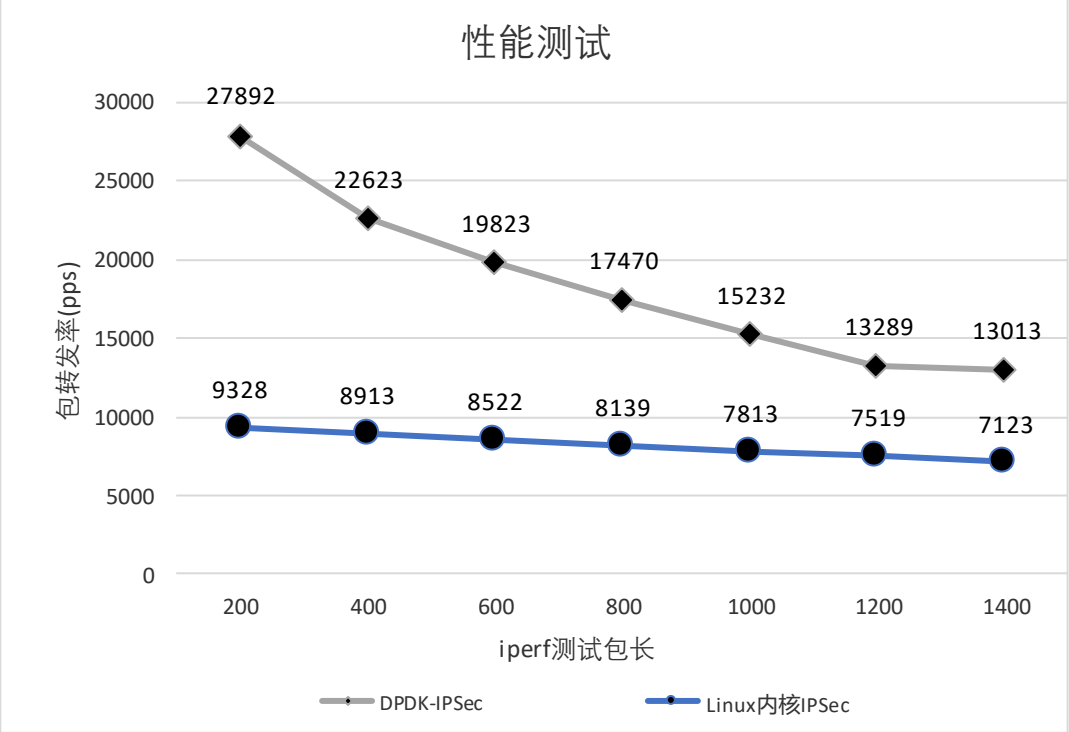

图 6-4 包长与包转发率测试

然后在服务端启动 nginx web 服务器，客户端使用 wget 下载 nginx 服务器上的大文件，可测出内核 IPSec VPN 和 DPDK IPSec VPN 使用 TCP 协议的吞吐量。吞吐量测试结果如图 6-4 所示。

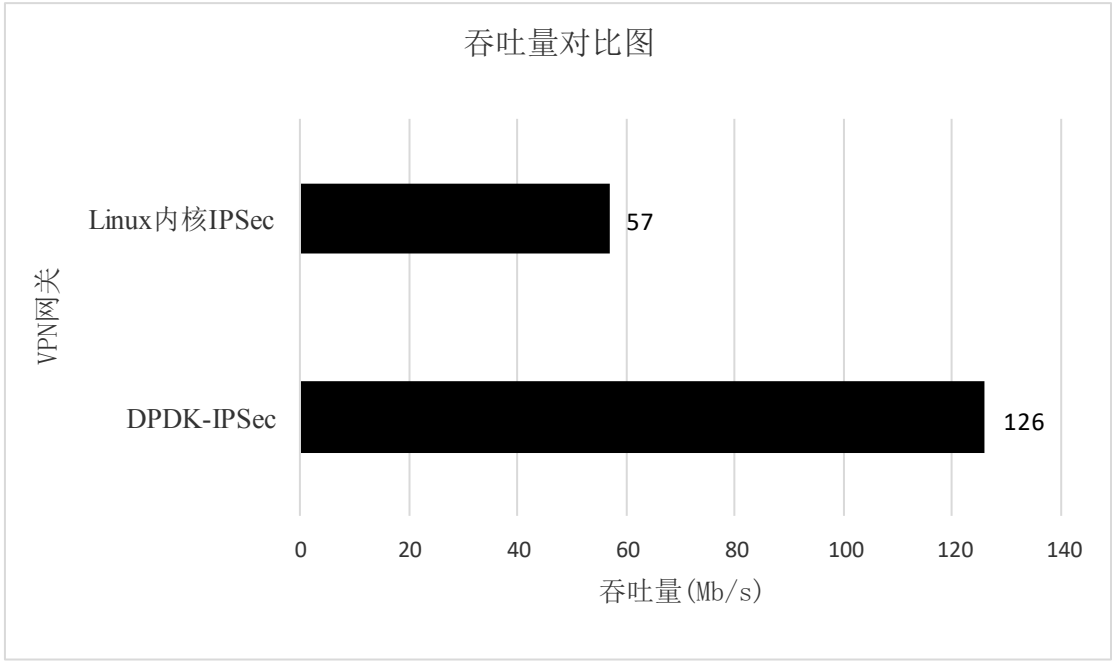

图 6-5 吞吐量对比

6.6 测试结论

本程序优点: 由图可得，本程序在小包处理方面优于内核 IPSec 网关，包转发率提升 3 倍，大包处理方面提升了 1.8 倍。包转发延迟更低，吞吐量提高了 2.2 倍。

本程序缺点:由于使用了轮训模式，使 CPU 一直处于满载状态。

6.7 本章小结

本章使用 iperf 对基于 DPDK 的 IPSec VPN 性能和传统 Linux 协议栈的 IPSec VPN 性能进行了测试。相比于传统协议栈，发现本程序在包转发率方面提升了 1.8 倍到 3 倍，在吞吐量方面提升了 2.2 倍。并经过压力测试，没有发生 coredump，发现本程序较为稳定。

结　论

由于传统内核协议栈过于冗余复杂导致数据包处理效率低下，本文使用 DPDK 技术在用户态协议栈实现了 IPSec VPN，使 VPN 在吞吐量，包延时和转发率方面又了较大提升，本论文主要完成了以下内容：

（1）列出 Linux 协议栈在数据拷贝，中断，锁竞争等方面存在的不足，并总结了 DPDK 的解决方案。使用 DPDK 框架设计出 IPSec VPN。采用 DPDK 轮训的方式处理报文，自行设计精小的用户态协议栈。

（2）本程序巧妙使用 KNI 网卡技术将 IKE 数据包转发给内核，由应用程序完成 IKE 并使用 Libnl 监听发给内核的会话密钥，使用多进程共享的方式将监听到会话密钥传给 DPDK 程序。

（3）本程序使用自行设计出 NAT 算法。数据经过解包后直接使用 NAT 处理，避免了传统流程中间的数据拷贝，中断等操作。经过测试效率有较大提升。

（4）本文实现了 ARP 协议，可以请求目标 IP 的主机的 MAC 地址，也可以处理其他主机发来的 ARP 请求。并可以通过 ARP 表封装数据包以太网头。

（5）本程序使用自己设计的协议处理 ESP 数据包，使用 DPDK 加密函数来完成加密和认证操作，SP 根据规则处理数据包，使用 SA 来管理会话密钥。

本文使用 DPDK 实现了高性能 IPSec VPN，通过测试，发现数据包在转发率上性能提升了 1.8 倍到 3 倍。在吞吐量上提升了 2.2 倍。在延迟和延迟抖动方面有很好的表现。根据在实现和研究中存在的问题，未来继续在以下方面进行研究：

（1）使用硬件加密加速对性能进行优化。本程序未使用 QAT 加密卡或 AES-NI 指令集对加密进行加速，因此加密方面还存在瓶颈。

（2）数据包的校验和计算仍通过软件算法实现。通过对 ShadowVPN 进行研究发现，经过 NAT 后，可通过数据包的端口或 IP 变化，只需要计算其差值，可使用原校验和进行计算，即可得到正确的校验和。也可以使用支持校验和 Offload 的网卡，通过启动网卡 Offload 通过硬件的方式对数据包的校验和重新进行计算。

（3）目前实现的 NAT 还无法实现分片处理。由于 IP 头分片后，除第一个数据包，新的数据包中不包含端口等信息，因此只能将分片后的数据包存储起来，等数据包全部到达重组后载进行 NAT 操作。此过程比较复杂，且对内存消耗比较大。

（4）目前仍不支持计费管理，QoS 限速，负载均衡和路由表等功能。随后可以通过配合 Radius 和 QoS 实现流量计费和限速等功能，并配合负载均衡算法和路

由表将接受的数据根据算法均衡的分配到指定的上级网关。

（5）目前仍未实现 AH 协议，随后可以实现 AH 协议。

（6）目前程序还不能实现 NAT 穿越。由于 ESP 数据包中不包含端口等信息，因此无法实现 NAT，可以通过使用 IPSec NAT-T 穿越技术，将 ESP 数据包封装在 UDP 协议中，使用 UDP 的 4500 端口进行通信，使得 NAT 处理经过 UDP 封装处理后的 ESP 数据包类似处理普通的 ESP 数据包。

参考文献

[1] 孙云霄. 面向企业用户的高性能 VPN 系统的设计与实现[D]. 哈尔滨：哈尔滨工业大学,2015.

[2] 王冠群. IPSec 中间人攻击检测方法与防御策略[D]. 哈尔滨：哈尔滨工业大学,2018.

[3] 吴承. 用户态 IPSec 协议栈的研究与实现[D]. 西安：西安电子科技大学,2014.

[4] 廖悦欣. IPSec 协议实现技术研究[D].广东：华南理工大学,2013.

[5] 穆瑞超. 基于 DPDK 的高性能 VPN 网关的研究与实现[D]. 哈尔滨：哈尔滨工业大学,2017.

[6] David Ely, Stefan Savage, and David Wetherall. Alpine: A User-Level Infrastructure for Network Protocol Development. Department of Computer Science and Engineerin University ofWashington, Seattle WA.

[7] Chris Maeda,Brian Bershad. Protocol Service Decomposition for High. PerformanceNetworking[C].Proceedings of the Fourteenth ACM Symposium on Operating Sy,;terms.

[8] Principles(SOSP).Asheville，NC，USA:1993:1263—1271P.

[9] Engelke E. WATTCP[J]. Home page at http://www. wattcp. com/index. shtml, 2004.

[10] Solarflare. http://www.openonload.org/.

[11] RFC4301, Security Architecture for the Internet Protocol [S]. USA: S. Kent,K. Seo 2015.12.

[12] RFC3602, The AES-CBC Cipher Algorithm and Its Use with IPsec [S]. USA: S. Frankel,R. Glenn,NIST,S. Kelly,Airespace 2003,9.

[13] RFC4303, IP Encapsulating Security Payload (ESP) [S]. USA: S. Kent. 2015.12.

[14] RFC4303, IP Encapsulating Security Payload (ESP) [S]. USA: S. Kent. 2015.12.

[15] RFC2367, PF_KEY Key Management API, Version 2 [S]. USA: D. McDonald,C. Metz,B. Phan 1998,7.

[16] 阚闯,栾新,戚玮玮.基于 Linux 的 XFRM 框架下 IPSec VPN 的研究[J].计算机工程,2008(20):109-111.

[17] 朱河清. 深入浅出 DPDK[M]. 机械工业出版社. 2016.

[18] 唐宏,柴桌原,任平,王勇.DPDK 应用基础[J].电信科学,2017,33(S1):268.

[19] 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(01):46-52.

[20]孙江,兰巨龙,李高鹏.Linux 环境下普适性零拷贝平台的研究与实现[J].计算机应用研究,2011,28(07):2621-2624.

[21]吕民强,吕丹丹.一种高效的零拷贝报文捕获系统[J].航空计算技术,2015,45(05):102-105.

[22]王婕,王健.基于 TOE 技术的嵌入式以太网接口设计[J].工业仪表与自动化装置,2011(06):34-36+41.

[23]周敬利,王志华,姜明华,徐漾,余胜生.基于 TCP/IP 卸载引擎的千兆网卡[J].计算机工程,2004(04):86-87+109.

[24]王小峰,时向泉,苏金树.一种 TCP/IP 卸载的数据零拷贝传输方法[J].计算机工程与科学,2008(02):135-138.

致　谢

光阴似箭，转眼间四年大学生活就要结束了。我即将离开校园，开始工作。还记得四年前我还是个懵懂的少年，对大学生活充满了无限向往与期待。大学四年间参加了王老师的实验室，在实验室学长和老师的指导下，我不仅学到了很多关于 VPN 的原理和编程，也学到了很多做人的道理。

感谢孙云霄老师在学术方面对我的指导与帮助，霄哥技艺精湛，对事要求严格，对人和蔼可亲，对在他的指引下我的能力不断提升。

感谢我的女朋友尹文琪，每当我遇到挫折难过的时候，她总会陪在我身边，给我鼓励，使我有了奋斗的动力，让我看到了未来的希望。往后余生，永远是你。

感谢父母对我的多年养育，总是向您索取却不曾说谢谢你，直到长大以后才懂得你不容易。

感谢王佰玲教授和刘扬老师在做人做事方面对我的教导和帮助。

感谢王冠群学姐在生活方面对我的指导与帮助。让我树立正确的人生观和价值观。感谢穆瑞超学长，傅春乐学长在技术上给予我的帮助。

感谢华为的同事和华为的师傅吴海滨。虽然与你们失之交臂，但是你们帮我迈出走向社会的第一步，并且在工作和生活上给予我无微不至的关怀，是我人生中至关重要的贵人。

感谢我的战友朱江辉同学，学习的时候一起拼搏，找工作的时候一起努力，实习的时候一起奋战。

最后，诚挚的感谢论文评阅人和各位专家教授对本文提出的宝贵意见。