electronics

MDPI

Article

A Generic High-Performance Architecture for VPN Gateways

Chunle Fu [1], Bailing Wang [1,*], Wei Wang [1], Ruichao Mu [1], Yunxiao Sun [1], Guodong Xin [1] and Yongzheng Zhang [2]

[1] School of Computer Science and Technology, Harbin Institute of Technology, Weihai 264200, China; chunle.fu@gmail.com (C.F.); wwhit@hit.edu.cn (W.W.); muruichao@kylinos.cn (R.M.); syx@hitwh.edu.cn (Y.S.); gdxin@hit.edu.cn (G.X.)

[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China; zhangyongzheng@iie.ac.cn

* Correspondence: wbl@hit.edu.cn

Abstract: Virtual private network (VPN) gateways are widely applied to provide secure end-to-end remote access and to relay reliable interconnected communication in cloud computing. As network convergence nodes, the performance of VPN gateways is limited by traditional methods of packet receiving and sending, the kernel protocol stack and the virtual network interface card. This paper proposes a generic high-performance architecture (GHPA) for VPN gateways in consideration of its generality and performance. In terms of generality, we redesign a generic VPN core framework by modeling a generic VPN communication model, formulating generic VPN core technologies and presenting corresponding core algorithms. In terms of performance, we propose a three-layer GHPA for VPN gateways by designing a VPN packet processing layer based on a data plane development kit (DPDK), implementing a user space basic protocol stack and applying our proposed generic VPN core framework. On the basis of the research work above, we implement a high-performance VPN (HP-VPN) and a traditional VPN (T-VPN) that complies with GHPA and traditional methods, respectively. Experimental results prove that the performance of HP-VPN based on GHPA is superior to T-VPN and other common VPNs in RTT, system throughput, packet forwarding rate and jitter. In addition, GHPA is extensible and applicable for other VPN gateways to improve their performance.

Keywords: cloud computing; VPN gateways; VPN communication model; data plane development kit (DPDK); generic high-performance architecture (GHPA)

check for updates

Citation: Fu, C.; Wang, B.; Wang, W.; Mu, R.; Sun, Y.; Xin, G.; Zhang, Y. A Generic High-Performance Architecture for VPN Gateways. *Electronics* **2024**, *13*, 2031. https://doi.org/10.3390/electronics13112031

Academic Editor: Dah-Jye Lee

Received: 6 May 2024
Revised: 19 May 2024
Accepted: 21 May 2024
Published: 23 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Cloud computing migrates traditional local computing to up-to-date remote computing to bring benefits of accessing large-scale storage resources, centralized computing resources and abundant service resources [1,2]. Meanwhile, frequent data downloading and uploading of cloud services brings enormous security challenges to both cloud users and cloud service providers, such as identity theft, data leakage and service hijacking [3–5]. To solve this problem, VPN technology is widely applied to guarantee reliable and stable communication between clients and clouds [6] or among interconnected clouds [7,8].

Traditional VPNs usually work in the form of software service in stationary devices, such as computers, routers, gateways and servers [9]. When directly applied as a software gateway in a network convergence node under the circumstances of cloud computing, the quality of the VPN service is usually unsatisfactory, leading to low-throughput, high-latency and unstable VPN connections. Existing improvements of network performance range from hardware-accelerated VPN gateways to optimizations of software VPN gateways. Hardware-accelerated VPN gateways usually rely on dedicated devices [10] or drivers [11,12], which is inextensible and inapplicable for various scenarios. Meanwhile, the existing optimizations of software VPN gateways are fragmentary because they do not take generality and high-performance into consideration simultaneously [13].

To improve the performance of VPN gateways on the premise of their generality, this paper researches a generic performance improvement method for VPN gateways. From the perspective of generality, we first propose a graph-based generic VPN communication model to abstract communication entities, links and, more importantly, the core duties of a VPN gateway. Then, we formulate three generic VPN core technologies: VPN session, VPN routing and VPN NAT, and propose their core algorithms: the VPN session matching algorithm, VPN routing searching algorithm and VPN NAT algorithm, respectively. Finally, we re-design a generic VPN core framework based on the work above. From the perspective of performance, we analyze the fundamentals of VPN gateways and clarify three performance bottlenecks: packet receiving and sending, the kernel protocol stack and the virtual network interface card. Then, we propose a three-layer generic high-performance architecture to improve the three bottlenecks above, respectively, including a DPDK-based [14] VPN packet processing layer, user space basic protocol stack and user space VPN NAT. In addition, we implement two prototype systems based on the proposed GHPA and traditional methods, respectively. The experimental results prove our proposed GHPA is superior to other traditional implementations in RTT, system throughput, packet forwarding rate and jitter. Meanwhile, GHPA is extensible and applicable to other VPN gateways.

Our main contributions in this paper are as follows:

- Formulation of a generic VPN communication model with three VPN core components, including a VPN session, route and NAT.
- Refactor of a DPDK-based generic high-performance architecture (GHPA) for VPN gateways.
- Implementations of a user space VPN gateway based on GHPA and a kernel-space VPN gateway, separately.
- Performance evaluations of software VPN gateways.

The rest of this paper is organized as follows. Section 2 provides the related work. Section 3 introduces fundamentals and performance bottlenecks of VPN gateways. Section 4 describes the modeling, formulation and algorithms of a proposed generic VPN core framework and the proposed GHPA for VPN gateways. Section 5 represents two prototype systems and then Section 6 introduces performance tests on our prototype systems and other common VPNs. Section 7 concludes this paper. All acronyms mentioned in this paper are shown in Table A1.

2. Related Work

Research on VPN gateways can be classified into three categories: solutions to network interconnections [15–17], assurance of security issues [18,19] and evaluations and improvements of network performance. Evaluations and improvements of network performance range from evaluations of VPN gateways [20–24] and hardware-accelerated VPN gateways [11,12,25,26] to improvements of software VPN gateways [13,27–29].

With respect to evaluations of VPN gateways, Zakaria et al. [20] examine the current deployment of VPN connections in IoT gateways, discussing their characteristics, benefits and drawbacks. Lawas et al. [21] evaluate the performance of SSTP and IKEv2 protocols by measuring throughput, jitter and delay. It is found that IKEv2 has significantly better performance than SSTP in their test-bed environment. Redzovic et al. [22] test the performance of an IPsec VPN router implemented by Quagga (https://goo.gl/NXbOfL accessed on 20 May 2024) and Strongswan (https://www.strongswan.org/ accessed on 1 October 2016) open-source software tools, and provide optimal VPN configuration according to their test results. Ismoyo and Wardhani [23] carry out performance tests on OpenVPN gateways with the ATHS3 block cipher algorithm and VEA stream cipher algorithm to confirm that the ATHS3 algorithm has better performance and efficiency. Kotuliak et al. [24] conduct comparative performance tests on OpenVPN and IPsec VPN under the conditions of different encryption algorithms. The test results show that OpenVPN and IPsec VPN have their own priorities in different situations.

In terms of hardware-accelerated VPN gateways, Yi et al. [25] propose a GPU acceleration framework to achieve both high generality and throughput under skewed flow size distributions. Raumer et al. [26] implement a VPN security gateway application, based on MoonGen and IPsec ESP tunnel mode, to demonstrate and evaluate the VPN throughput and energy saving performance of the improved device driver. Heinemann et al. [12] utilize a GPU to improve the encryption and decryption efficiency of IPsec VPN, in which the GPU achieves encryption and decryption 2.9 to 8.7 times faster than a CPU. Turan et al. [11] design a hardware-based coprocessor to accelerate the cryptographic operations of an open-source software VPN called SigmaVPN, which reduces time overhead in encryption by 93% compared to the software-only solution and increases TCP and UDP bandwidth by a factor of 4.36 and 5.36, respectively, for 1024-byte Ethernet frames.

In terms of improvements in software VPN gateways, Wei et al. [27] propose a scheme of adaptively adjusting the size of the thread pool based on a self-learning algorithm to improve the processing performance of a VPN gateway. Pudelko et al. [13] evaluate existing open-source VPN projects, OpenVPN, Linux IPsec, and WireGuard, by implementing WireGuard-compatible VPN benchmarking with open-source solutions. Then, they proposed a WireGuard pipeline architecture on top of DPDK, achieving 6.2 Mpps and 40 Gbit/s, which is the fastest of all the evaluated VPN implementations. Li [28] establish a high-performance software router based on a DPDK framework to approach the line rate of our 40 Gb network interface card (NIC). Zhang et al. [29] propose an IPsec thumbnail protocol to implement IPsec acceleration in a pure software method, which achieves obvious improvements in IPsec transmission speed.

The most relevant pieces of research for this paper are [13,28], which similarly utilize a DPDK packet processing platform to overcome the performance bottlenecks of Linux kernel and NIC [30]. Related user space packet processing frameworks are netmap [31] and PF RING ZC [32]. DPDK has been proven to show better performance in throughput and latency [33,34]. Therefore, our proposed GHPA is based on DPDK. Limitations of the research in [13] are implemented by open-source projects and their implementation is a testbed for different software architectures, not a VPN gateway meant for production use. In addition, Li [28] verifies that DPDK-based software routers are able to approach the line rate of NIC, not in relation to VPNs.

To summarize, the existing improvements in software VPN gateways are fragmentary because they do not take generality and high performance into consideration simultaneously. Therefore, this paper conducts comprehensive research on VPN gateways by formulating a VPN core communication model, modularizing core components of VPN sessions, routes and NAT and refactoring a DPDK-based generic high-performance architecture.

3. Background

In this section, we first discuss the fundamentals of traditional VPN gateways. Then, performance bottlenecks of traditional VPN gateways are analyzed according to their implementations and working mechanisms. Finally, our research work is clarified to solve the existing performance bottlenecks.

3.1. Introduction to Fundamentals of Traditional VPN Gateways

This section discusses the fundamentals of traditional VPN gateways from the perspectives of classification, working mechanisms and performance evaluation standards.

3.1.1. Classification

VPN gateways can be classified into different types according to different classification standards. From a gateway type perspective, VPN gateways can be divided into hardware VPN gateways and software VPN gateways. Hardware VPN gateways are usually customizable devices equipped with hardware accelerators to provide a high-performance VPN service for large-scale enterprises or governments. In contrast, software VPN gateways work in the form of software services on general servers or gateway devices. Using

network topologies as a classification standard, VPN gateways can be categorized into centralized gateways and peer-to-peer (P2P) gateways. The former ones comply to a client/server (C/S) service mode by acting as centralized switches in VPN topologies while the latter ones directly communicate with other gateways. Considering different protocols applied by VPN gateways, VPN gateways can be classified into PPTP VPN gateways, L2TP VPN gateways, IPsec VPN gateways, SSL VPN gateways and other VPN gateways. This paper researches the performance bottlenecks of centralized software VPN gateways and provides a generic high-performance architecture for VPN gateways (which refers to centralized software VPN gateways for the remainder of this paper).

3.1.2. Working Mechanisms

To better understand the working mechanisms of VPN gateways, we first discuss the core duties of VPN gateways in a VPN topology from a packet's perspective. A VPN gateway mainly processes two kinds of packets: VPN packets from VPN clients and normal packets from target servers. A VPN packet encapsulates an inner packet in an outer packet. The inner packet, carrying a real data payload of the applications of VPN clients, is encrypted and encapsulated as the VPN payload of the outer packet. A normal packet is a standard packet that carries either requests from VPN clients to target servers or responses from target servers back to VPN clients. Therefore, a VPN gateway plays the role of a bridge in VPN topological structure by delivering packets to destinations, ranging from VPN clients inside its virtual network to target servers outside its virtual network. The former mode is applied for interconnecting hosts from different local area networks (LANs), so we name it the relay mode. The latter mode is called the agent mode because a VPN gateway forwards requests from VPN clients to target servers and delivers responses from target servers back to VPN clients.

Figure 1 depicts the working mechanism of a traditional VPN gateway both in relay mode and agent mode. Traditional VPN gateways receive VPN packets through an NIC driver, kernel protocol stack, and Linux socket, sequentially. Then, the VPN gateway process decrypts the VPN payload with a corresponding session key for an inner packet and checks its destination address. If the address is one of the other VPN clients, the VPN gateway works in relay mode by encrypting the inner packet with the peer session key and sending it to the peer VPN client. If not, the inner packet is written to the virtual network interface card (VNIC) by a character-driven device. Afterwards, the packet is routed to the kernel protocol stack to execute source network address translation (SNAT) and sent to the target server.

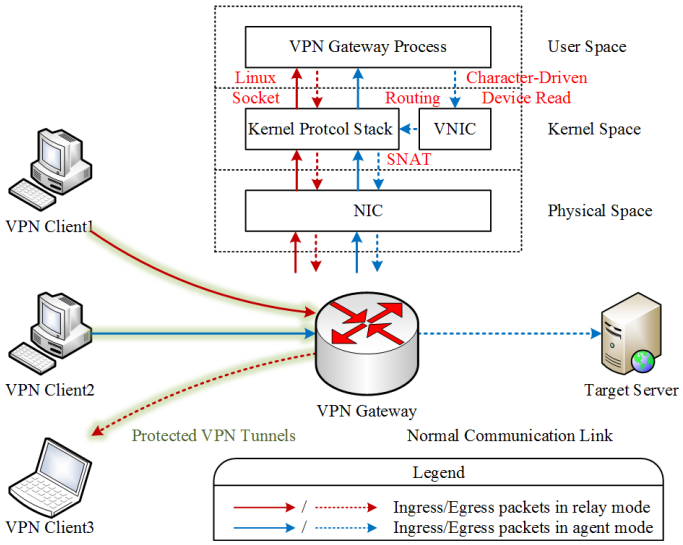

Figure 1. Working mechanism of a VPN gateway both in relay mode and agent mode.

3.1.3. Performance Evaluation Standards

In general, the performance of a network application is measured by round-trip time (RTT), system throughput, packet forwarding rate and jitter. Considering that the protocols of inner packets in VPN tunnels are likely to affect the performance of VPN tunnels, the system throughput of a VPN gateway should be evaluated according to different transmission protocols: UDP or TCP. Therefore, performance evaluation standards of VPN gateways include RTT, UDP throughput (referring to system throughput via UDP transmission), UDP packet forwarding rate, UDP jitter and TCP throughput (referring to system throughput via TCP transmission). Generally speaking, a high-performance VPN gateway is supposed to provide a high-performance VPN service with low latency, high throughput, a high packet forwarding rate and low jitter.

3.2. Analysis of Performance Bottlenecks of Traditional VPN Gateways

This section analyzes three performance bottlenecks of traditional VPN gateways: packet receiving and sending, the kernel protocol stack and VNIC. The details are as follows.

3.2.1. Packet Receiving and Sending

As shown in Figure 1, the packet receiving and sending of traditional VPN gateways is implemented based on a Linux socket. Taking VPN gateways receiving packets as an example, the whole process of packet receiving can be divided into three stages: receiving NIC and its driver, kernel protocol stack decoding, and VPN gateway process receiving. When NIC receives ingress packets and triggers hardware interrupts, the NIC driver copies the packets from RX queues to ring buffer in kernel space. The kernel protocol stack then performs protocol validation, firewall filtering, IP routing and copies the packets from ring buffer to socket buffer by software interrupts. Finally, the VPN gateway process receives the packets from socket buffers.

The process of packet receiving and sending causes system interrupts and memory copies several times. Because of the resource management of operating systems, expensive system calls, context switches and memory allocation and release occur with the advent of system interrupts and memory copies. When massive packets arrive in NIC, the situation is worse in that packets are discarded by NIC due to the exhaustion of system resources. Therefore, the traditional Linux socket method of packet receiving and sending is one of the most important performance bottlenecks of VPN gateways.

3.2.2. Kernel Protocol Stack

A kernel protocol stack mainly functions as both a protocol decoder and a protocol encoder that meets the OSI seven-layer architecture. In addition, it provides IP routing and Netfilter services for VPN gateways. IP routing is used to route ingress packets from the kernel protocol stack to upper VPN gateway processes and to route egress packets from VPN gateway processes to VNIC, which is necessary for both relay-mode VPN gateways and agent-mode VPN gateways. The Netfilter service, applied to execute SNAT for egress packets and destination NAT (DNAT) for ingress packets, is compulsory for agent-mode VPN gateways only.

In respect of its functions, the kernel protocol stack is abundant but redundant due to excessive memory consumption and computational overheads. In respect of its performance, the efficiency of passing a packet from user space to kernel space by VNIC and then executing NAT by the kernel Netfilter service is unsatisfactory. Hence, the kernel protocol stack becomes a performance bottleneck of VPN gateways in a high-speed network environment.

3.2.3. Virtual Network Interface Card

VNIC is widely used to establish a virtual network device in most implementations of agent-mode VPN gateways. VNIC mainly consists of two parts: a tun/tap device and character-driven device. The tun/tap device is a virtual network interface to read packets

from VPN gateway processes when sending a request to target servers and to write packets to VPN gateway processes when receiving responses from target servers. The character-driven device manipulates the procedure of packet reading and writing between user space and kernel space.

Indeed, VNIC simplifies the implementation of agent-mode VPN gateways by utilizing system IP routing and the NAT functions of the kernel Netfilter service. However, it increases packet processing time in kernel space and occupies more system resources when switching packets between user space and kernel space. In addition, the interfaces in packet reading and writing provided by the character-driven device are as vulnerable as the Linux socket under the circumstances of fast packet processing. Thus, VNIC is another performance bottleneck of VPN gateways.

3.3. Summary

Through this introduction to fundamentals and analysis of the performance bottlenecks of traditional VPN gateways, we find that the performance of VPN gateways is limited by traditional packet receiving and sending, the kernel protocol stack and VNIC. Therefore, our improvement work is to apply a DPDK-based fast packet processing method, to design a user space protocol stack and to implement user space NAT instead of using VNIC with the Netfiler service, respectively. To provide a generic high-performance architecture for VPN gateways, re-designing a generic VPN core framework to provide general VPN functions is equally important.

4. Proposed Generic High-Performance Architecture

Aiming to address the existing performance bottlenecks of traditional VPN gateways mentioned in Section 3.2, this section introduces a generic high-performance method for VPN gateways. We start by modeling a generic VPN communication model, formulating generic VPN technologies and designing corresponding core algorithms. Then, we present the proposed three-layer generic high-performance architecture (GHPA), which includes a DPDK-based VPN packet processing layer, a user space basic protocol stack and a generic VPN core framework. The symbols in this section are described in Table 1.

Table 1. Symbols and descriptions for VPN communication model, session, route and NAT.

Categories	Symbols	Descriptions
VPN communication model	G	VPN topology
	V	VPN communication entities
	E	VPN communication tunnel
	$\{v_i, v_j\}$	Underlay communication links between entity v_i and v_j
	$\{v_i, v_n, v_j\}$	Underlay communication links between entity v_i , v_n and v_j
VPN session	$\{a_{v_i}, p_{v_i}\}$	Network address and port for entities v_i
	$p_{e_i}^{(s)}$	Communication procedures of VPN tunnel e_i
	k_{e_i}	Session keys of VPN tunnel e_i
	S_{e_i}	Session information of VPN tunnel e_i
VPN route	R	VPN routing policies
VPN NAT	N	VPN NAT policies, including N^s for SNAT and N^d for DNAT

4.1. Generic VPN Communication Model

To re-design a generic VPN core framework, the duties of a generic VPN gateway are supposed to be clarified ahead so that a graph-based generic VPN communication model can be proposed to abstract communication entities in VPN topology.

As shown in Figure 2, this model is abstracted from a common VPN topology to a graph $G = (V, E)$. The vertex set of graph G represents $n + m$ communication entities (denoted by set V) in a VPN topology, which are composed of $n - 1$ VPN clients (denoted

by set V_c), a VPN gateway (denoted by set V_g) and m target servers (denoted by set V_s), as in Expressions (1) and (2):

$$V = V_c \cup V_g \cup V_s \tag{1}$$

$$v_i = \begin{cases} v_i \in V_c, & i \in [1, n-1] \\ v_i \in V_g, & i = n \\ v_i \in V_s, & i \in [n+1, n+m] \end{cases} \tag{2}$$

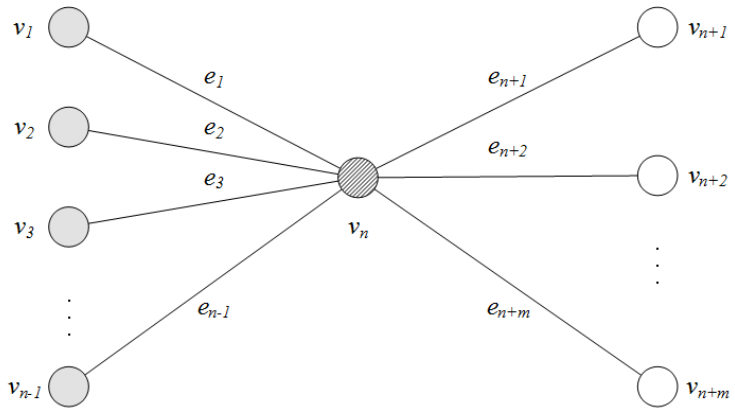

Figure 2. Graph-based generic VPN communication model.

The edge set $E = \{e_1, e_2, \ldots, v_{n+m}\}$ of graph G represents $n + m - 1$ communication links in VPN topology so that the connectivity of graph G can be expressed by the adjacency matrix (denoted by $A_{ij} = [a_{ij}]$) in Expression (3):

$$A_{ij} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 1 & \cdots & 1 & 0 & 1 & \cdots & 1 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} \tag{3}$$

A value of 0 refers to vertex unreachable while a value of 1 refers to vertex reachable. As all network traffic in a VPN topology is switched by a centralized VPN gateway, the paths of graph G can be expressed by sequence $\{v_i, v_j\}$ or $\{v_i, v_n, v_j\}$ as in Expressions (4) and (5):

$$\{v_i, v_j\} = \begin{cases} p_{(c,g)}, & if\ i \in [1, n-1],\ j = n \\ p_{(g,c)}, & if\ i = n, j \in [1, n-1] \\ \varnothing, & otherwise \end{cases} \tag{4}$$

$$\{v_i, v_n, v_j\} = \begin{cases} p_{(c,g,c)}, & if\ i \in [1, n-1],\ j \in [1, n-1] \\ p_{(c,g,s)}, & if\ i \in [1, n-1], j \in [n+1, n+m] \\ p_{(s,g,c)}, & if\ i \in [n+1, n+m], j \in [1, n-1] \\ \varnothing, & otherwise \end{cases} \tag{5}$$

Path $p_{(c,g)}$ and $p_{(g,\,c)}$ represent direct communication links between a VPN gateway and its clients. Path $p_{(c,g,c)}$ refers to a relay communication link via a VPN gateway. Path $p_{(c,g,s)}$ and $p_{(s,g,\,c)}$ are agent communication links via a VPN gateway. The connectivity of paths to a graph is as important as the management of communication links to a VPN

gateway. A VPN gateway mainly manages three kinds of communication links during its life cycle: handshake communication links (referring to path $p_{(c,g)}$ and $p_{(g,\,c)}$), relay-mode data communication links (referring to path $p_{(c,g,c)}$) and agent-mode data communication links (referring to path $p_{(c,g,s)}$ and $p_{(s,g,\,c)}$).

Through the abstraction of a generic VPN communication model, the duties of a generic VPN gateway are clarified to include handshake communication, relay-mode data communication and agent-mode data communication from the perspective of VPN communication procedures. Then, generic theoretical and algorithmic details of the three key points above will be explored in Sections 4.2 and 4.3, respectively.

4.2. Generic VPN Core Technologies

In this section, three VPN core technologies are introduced in formal language to support the three communication procedures of VPN gateways mentioned in Section 4.1, including VPN session, VPN routing and VPN NAT.

4.2.1. Formulation of VPN Session

A VPN session is used to process VPN packets during the lifecycle of a VPN tunnel, including the identification of VPN connections and drive of VPN protocols.

In terms of connection identification, a VPN tunnel is a TCP-based or UDP-based persistent connection maintained by a VPN gateway in public link, which can be identified by a unique 4-tuple: source network address, source port, destination network address and destination port. In particular, VPN gateways usually utilize fixed network addresses and ports to listen to requests from remote VPN clients so that a VPN connection (denoted by tuple c_{e_i}) can be simplified from a 4-tuple set to a 2-tuple set, as in Expression (6):

$$c_{e_i} = \{a_{v_i}, p_{v_i}, a_{v_n}, p_{v_n}\} = \{a_{v_i}, p_{v_i}\} \tag{6}$$

Element a_{v_i} and p_{v_i} refer to the network address and port of VPN client v_i, respectively.

The drive of VPN protocols is an equally important task for a VPN session. To formulate a protocol-independent VPN session, VPN communication procedures can be divided into two stages: handshake stages and transmission stages. In the handshake stage, VPN gateways utilize identity authentication protocols to authenticate VPN clients and key negotiation protocols to generate session keys for next stage use. Then, VPN gateways use the session keys to encrypt and decrypt packets in the transmission stage of either relay mode or agent mode. Therefore, the procedures of VPN connection e_i can be expressed by set $p_{e_i}^{(s)}$, as in Expression (7):

$$p_{e_i}^{(s)} = \begin{cases} p_{e_i}^{(1)}, & e_i \in p_{(c,g)}, p_{(g,c)} \\ p_{e_i}^{(2)}, & e_i \in p_{(c,g,c)} \\ p_{e_i}^{(3)}, & e_i \in p_{(c,g,s)}, p_{(s,g,c)} \end{cases} \tag{7}$$

In addition, the session keys of VPN connection e_i can be expressed by set k_{e_i}.

To summarize, a VPN session is used to store information about connections, communication procedures and session keys in a VPN tunnel, as in Expression (8):

$$S_{e_i} = \left\{c_{e_i}, p_{e_i}^{(s)}, k_{e_i}\right\}, s \in [1, 3] \tag{8}$$

As such, a VPN gateway manages a set of VPN session information (referring to set S) during its runtime, as in Expression (9):

$$S = \bigcup_{i=1}^{n-1} S_{e_i} \tag{9}$$

4.2.2. Formulation of VPN Routing

Differing from traditional network routing, VPN routing is required to find a session key of the peer VPN client according to its virtual address of an inner packet in relay-mode VPN gateways. After decrypting the payload of an ingress VPN packet, a VPN gateway decodes the inner packet to get its destination virtual address a'_{v_j}. Then, a'_{v_j} is utilized to find the session key k_{e_j} of communication link e_j. Afterward, the VPN gateway encrypts the inner packet with session key k_{e_j}, encapsulates the inner packet with a VPN header again and then sends it to the peer VPN client.

In short, a VPN routing policy is a 2-tuple (referring to tuple R_{e_j}) to associate a virtual address a'_{v_j} with the session key k_{e_j}, as in Expression (10):

$$R_{e_j} = \left\{ a'_{v_j}, k_{e_j}, a_{v_j}, \right\}, v_j \in V_c \tag{10}$$

Therefore, a VPN gateway manages a set of VPN routing policies (referring to set R) during its runtime, as in Expression (11):

$$R = \bigcup_{j=1}^{n-1} R_{e_j} \tag{11}$$

4.2.3. Formulation of VPN NAT

VPN NAT is applied to transfer a virtual address to a gateway network address and replace an application port with a gateway port for an inner packet in agent-mode VPN gateways. In essence, VPN NAT is a mapping strategy between applications of VPN clients and services of target servers. A VPN gateway executes SNAT for an inner request packet from a VPN client to a target server and executes DNAT for a response packet in the opposite direction. The two NAT strategies above can be expressed by mapping N_i^s and mapping N_j^d, as in Expressions (12) and (13):

$$N_i^s : \left\{ (a'_{v_i},\ p'_{v_i}, a_{v_j}, p_{v_j}, p_{ij}) \right\} \rightarrow \left\{ (a_{v_n},\ p'_{v_n}) \right\} \tag{12}$$

$$N_j^d : \left\{ (a_{v_j}, p_{v_j}, a_{v_n},\ p'_{v_n}, p_{ij}) \right\} \rightarrow \left\{ (a'_{v_i},\ p'_{v_i}) \right\} \tag{13}$$

The 2-tuple $(a'_{v_i},\ p'_{v_i})$ refers to the virtual address and application source port of VPN client v_i. The 2-tuple (a_{v_j}, p_{v_j}) refers to the destination network address and destination port of target server v_j. The 2-tuple $(a_{v_n},\ p'_{v_n})$ refers to the network address and available port of VPN gateway v_n. The element p_{ij} refers to the communication protocol between the applications of VPN client v_i and service of target server v_j.

In conclusion, a VPN gateway manages a set of SNAT and DNAT strategies (referring to set N) during its runtime, as in Expression (14):

$$N = \{ \bigcup_{i=1}^{n-1} N_i^s \} \cup \{ \bigcup_{j=1}^{n+m} N_j^d \} \tag{14}$$

4.3. Generic VPN Core Algorithms

On the basis of the description and formulation of the VPN session, VPN routing and VPN NAT, this section further discusses the core algorithms of these three technologies.

4.3.1. VPN Session Matching Algorithm

A VPN gateway manages a VPN session set S (referring to Expression (9)) of all connected clients and utilizes specific session information to drive a specific VPN tunnel for a specific client during its runtime. Therefore, we propose a session matching algorithm based on a chained hash table to efficiently match real-time VPN packets with VPN sessions.

A session hash table consists of two kinds of chains: a two-way hash chain that stores VPN session S_{e_i} and settles hash collision problems, and a free chain, which is a pre-allocated session chain to store new sessions without the overheads of dynamic memory allocation. The hash index of a hash bucket is computed by 2-tuple c_{e_x} with the crc32 hash algorithm. Figure 3 depicts the storage structure of the VPN session table.

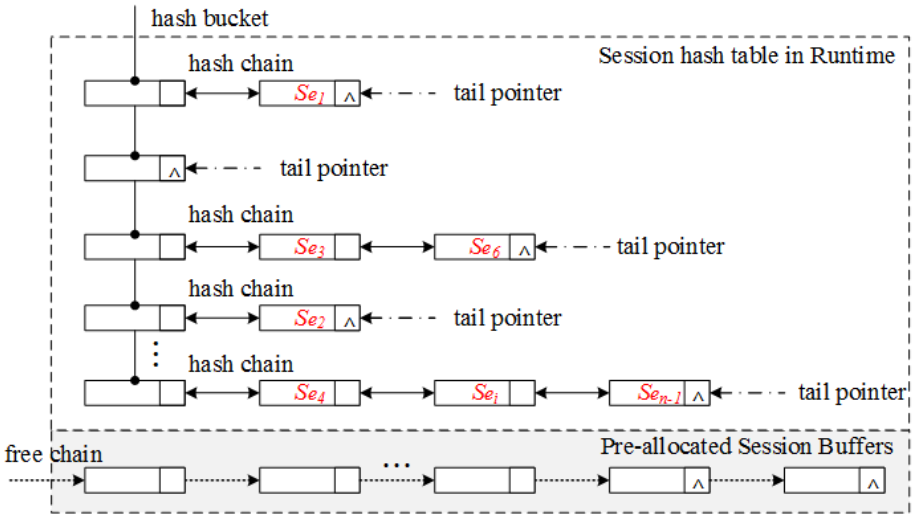

Figure 3. Storage structure of VPN session table.

Furthermore, our proposed VPN session matching algorithm, driven by communication procedures based on the chained hash table, is described in Algorithm 1. When receiving an ingress VPN packet, hash index i_x of its 2-tuple c_{e_x} is computed and the VPN session table is searched with ix and c_{e_x}. If no session node is found, an insertion process is activated. First, a free session node of the free chain is used to store information of S_{e_x} according to Expression (8). Then, hash index i_x is utilized to find the corresponding index of the hash bucket. Afterwards, session node S_{e_x} is successfully inserted in the tail of the hash chain and is pointed by the corresponding tail pointer. After finding or inserting a session node, a VPN session is matched with the ingress packet and then its VPN header is decoded to drive different communication procedures.

Algorithm 1 Algorithm to match VPN sessions.

Input:
The connection information of an ingress packet, c_{e_x};
The VPN session hash table, H_s;

```
compute hash index i_x with c_ex;
search S_ex in H_s with i_x and c_ex;
if S_ex is null then
    insert S_ex into H_s;
    initialize s ← 1;
end if
decode VPN header to get communication procedure s;
switch(s)
case 1:
    drive VPN handshake procedure p_ex^(1);
case 2:
    decrypt the packet payload with k_ex, k_ex ∈ S_ex;
    execute VPN routing searching with Algorithm 2;
    drive relay mode transmission procedure p_ex^(2);
```

```
15  case 3:
16      decrypt the packet payload with k_{e_x}, k_{e_x} ∈ S_{e_x};
17      execute VPN NAT with Algorithm 3;
18      drive agent mode transmission procedure p_{e_x}^{(2)};
19  default:
20      ignore invalid packets;
21  end switch
```

4.3.2. VPN Routing Searching Algorithm

A VPN gateway manages VPN routing policies set R (referring to Expression (11)) to find a specific session key for a specific virtual address in relay-mode data transmission. In view of network address storage and searching methods in the kernel routing table, we propose a VPN routing searching algorithm based on a Patricia tree to provide real-time matching for VPN routing policies.

A Patricia tree is a space-optimized trie applied in the traditional network routing algorithm of 4.4 BSD-Lite Linux kernel. It is a binary tree with two types of nodes: internal nodes and leaf nodes. Each node stores information of a bit sequence, a comparative bit and pointers pointing to its parent node and child nodes, while only leaf nodes store information of keywords. Therefore, the storage structure of our proposed VPN routing table is shown in Figure 4, which takes storing three routing policies R_{e_2}, R_{e_3} and R_{e_4} as an example. Parent nodes are split by the value of comparative bits, splitting to the left child and child node while the value of each comparative bit equals to 0 and 1, respectively.

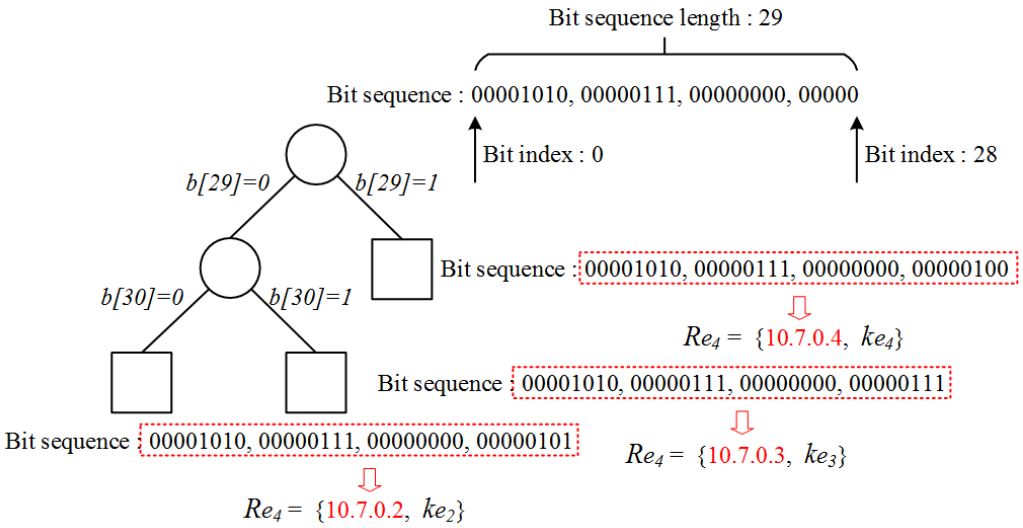

Figure 4. Storage structure of VPN routing table.

Hence, our proposed VPN routing searching algorithm is described in Algorithm 2. The whole process can be divided into three steps. First, an address a in bit sequence format is transferred from a string formatted virtual address a'_{v_x}. Second, the routing table T is searched from its root node to its leaf node by comparative bit recursively. Third, a is compared with address b of the leaf node and the session key of address a'_{v_x} is obtained.

Algorithm 2 Algorithm to search VPN routing policies.

Input:
The destination addresses of an inner packet, a'_{v_x};
The VPN routing table, T;
Output:
The session key of communication link e_x, k_{e_x};

```
if T is null then
    return null;
end if
transfer a'_{v_x} to bit sequence a;
let T' points to root node of T;
while T' is not leaf node do
    read comparative bit i of T';
    if a[i] == 0 then
        let T' points to left child of T';
    else
        let T' points to right child of T';
    end if
end while
read bit sequence b of T';
if b equals to a then
    return k_{e_x} of T';
else
    return null;
end if
```

4.3.3. VPN NAT Algorithm

A VPN gateway manages VPN NAT strategies set N (referring to Expression (14)) to execute SNAT for egress packets and execute DNAT for ingress packets in agent-mode data transmission. We propose a VPN SNAT algorithm and a VPN DNAT algorithm based on chained hash tables.

VPN NAT hash tables consist of a SNAT table N^s and a DNAT table N^d. Each table has the same two chains and same hash function as session hash tables. In addition, table Ns and Nd share a two-way time chain. Figure 5 illustrates the storage structures of two NAT tables, especially omitting the less important tail pointers and emphasizing the structure of the shared two-way time chain. Algorithm 3 describes the process of executing SNAT for egress packets, which can be interpreted as inserting SNAT and DNAT strategies for later packets' use. Similarly, the process of executing DNAT for ingress packets is easier, which can be considered as searching DNAT strategies. As such, the details of the VPN DNAT algorithm will not be discussed any further.

Compared to inserting and searching NAT strategies, removing these strategies is relatively difficult for a VPN gateway. As a SNAT strategy N_i^s and a DNAT strategy N_i^d are inserted in pairs (N_i^s, N_i^d), we utilize a time chain to connect all the strategy pairs. Therefore, a VPN gateway can remove timeout NAT strategy pairs by periodically checking the shared time chain.

Algorithm 3 Algorithm to execute SNAT for egress packets.

Input:
The 5-tuple of an egress packet, $(a_x,\ p_x,\ a_y,\ p_y,\ p_{xy})$;
The SNAT strategies set referring to Expression (12), N^s;
The DNAT strategies set referring to Expression (13), N^d;

```
if (a_x, p_x, a_y, p_y, p_xy) is null then
    return null;
end if
if N^s(a_x, p_x, a_y, p_y, p_xy) is not null then
    use N^s(a_x, p_x, a_y, p_y, p_xy) to do SNAT for the packet;
else
    find available 2-tuple (a_z, p_z) of the VPN gateway;
    add strategy (a_x, p_x, a_y, p_y, p_xy) → (a_z, p_z) to N^s;
    add strategy (a_y, p_y, a_z, p_z, p_xy) → (a_x, p_x) to N^d;
    use (a_z, p_z) to do SNAT for the packet;
end if
```

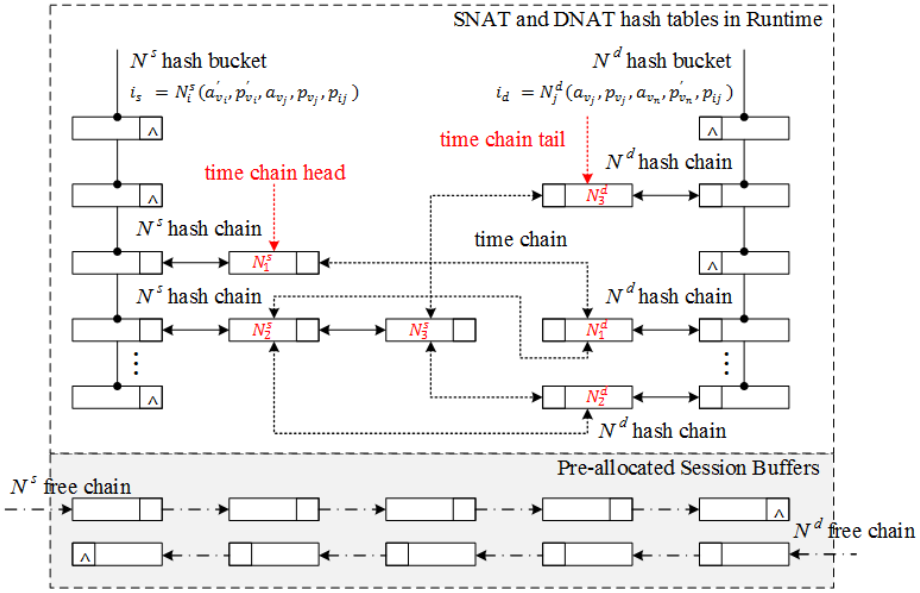

Figure 5. Storage structure of VPN NAT tables.

4.4. Proposed Generic High-Performance Architecture

This section introduces our proposed generic high-performance architecture (HGPA) in light of the existing performance bottlenecks of VPN gateways. Figure 6 presents the overall architecture of the three-layer GHPA. On the bottom layer, a DPDK-based VPN packet processing layer is proposed to replace the traditional way of using Linux socket to improve the existing bottleneck of packet receiving and sending mentioned in Section 3.2.1. On the middle layer, a user space basic protocol stack is designed and applied instead of using the redundant and inefficient kernel protocol stack referred to in Section 3.2.2. In addition, a generic VPN core framework on the top layer is presented to enclose the generic core technologies and algorithms of VPN gateways, in which a user space NAT implementation avoids the bottlenecks of VNIC and the kernel Netfilter service according to Section 3.2.3. The details of each layer are discussed as follows.

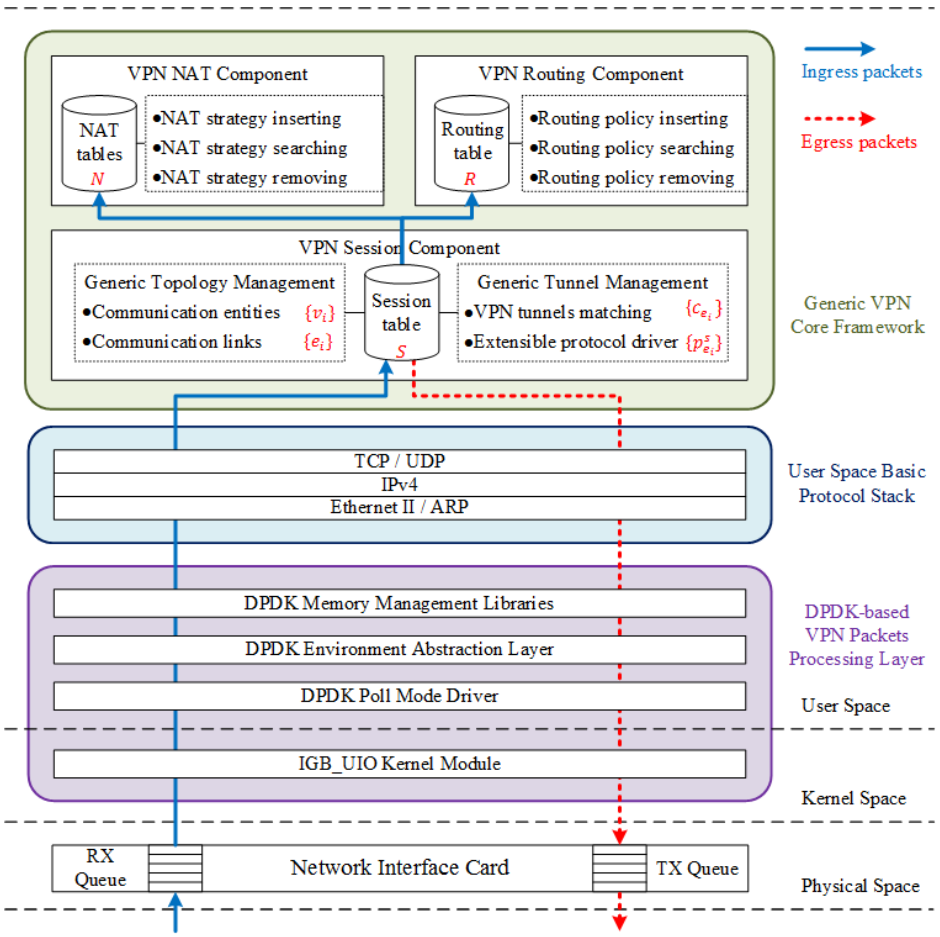

Figure 6. Overall architecture of GHPA.

4.4.1. DPDK-Based VPN Packet Processing Layer

The design of the DPDK-based VPN packet processing layer complies with DPDK arts and crafts for fast packet receiving and sending between the kernel space and user space. This layer is mainly composed of four parts: an IGB_UIO kernel module, DPDK PMD, DPDK EAL and DPDK memory management libraries. The IGB_UIO kernel module takes over control of NIC in kernel space by unbinding the traditional IGB kernel module and registering the NIC device. DPDK PMD can directly access packets from the RX and TX descriptors without any interrupts to quickly receive and send packets in user space. DPDK EAL provides an easy-to-use programming interface, which is utilized to load and launch the DPDK working environment, reserve system memory and configure and initialize NIC. Library Ring, Mempool and Mbuf make up the hardcore of DPDK memory management libraries. Among them, library Mbuf is a basic data structure to store link layer packets to be received by upper layers or to be sent by NIC.

4.4.2. User Space Basic Protocol Stack

Packets received by the upper packet processing layer are unable to be directly used by upper applications. Therefore, the user space basic protocol stack is born to bridge the gap between packet processing and upper applications. Meanwhile, the inefficient kernel protocol stack is abandoned in this implementation method. This basic protocol stack provides protocol validation, packet encapsulation, packet decapsulation and checksum computing, which can be regarded as a minimal subset of the kernel protocol stack on the premise of implementing all core functions of a VPN gateway. As shown in Figure 6, Ethernet II, an address resolution protocol (ARP), IPv4, TCP and UDP are required to guarantee normal communication between a VPN gateway and its clients both in relay

mode and agent mode. Ethernet II, the carrier of upper protocols, plays a vital role in packet delivery. To ensure network reachability during the work cycle of a VPN gateway, ARP is utilized to periodically notify the physical address and network address of a VPN gateway to its defaulted gateway device. In addition, there is no doubt that IPv4, TCP and UDP are essential to carry upper VPN protocols in a public network link.

4.4.3. Generic VPN Core Framework

A generic VPN core framework is re-designed on the basis of the modeling, formulation and algorithms design of the generic VPN core technologies from Sections 4.1–4.3. As shown in Figure 6, this framework mainly covers three components: the VPN session component, VPN routing component and VPN NAT component, which are compulsory to guarantee the core functions of traditional VPN gateways. The VPN session component provides generic topology management and generic tunnel management for VPN gateways. The former refers to monitoring and managing the real-time work state of communication entities (referring to set $\{v_i\}$) and communication links (referring to set $\{e_i\}$). The latter refers to matching VPN sessions (referring to set S) in real-time and driving communication procedures (referring to set $\{p_{e_i}^{(s)}\}$) according to different VPN protocols, which is extensible and applicable for other common VPNs. The VPN routing component is utilized for the management of VPN routing policies (referring to set R) and is applied exclusively to meet the demands of data transmission via relay-mode VPN gateways, such as interconnecting hosts from different LANs and building interconnected private clouds through public network links. The VPN NAT component is specialized for managing VPN NAT strategies for agent-mode VPN gateways, and is used for secure remote access and bypassing network censorship. In addition, the VPN NAT component provides a highly efficient method for executing SNAT for egress packets and DNAT for ingress packets in user space rather than combining VNIC with the kernel Netfilter service to implement NAT, as in traditional methods.

5. Prototype Systems

This section introduces two prototype systems in our research work: a high-performance VPN (HP-VPN) gateway based on GHPA and a traditional VPN (T-VPN) gateway based on traditional Linux socket. The similarities and differences of the two prototype systems are discussed as follow.

5.1. High-Performance VPN (HP-VPN) Gateway Based on GHPA

The implementation of HP-VPN strictly complies with our proposed GHPA. As the core functions of VPN gateways have been provided by GHPA, HP-VPN is relatively easy to implement as long as we design a set of VPN communication protocols to meet our actual demands. The protocols of HP-VPN are designed based on the libsodium crypto library, which is a portable, cross-compliable and installable fork of the Networking and Cryptography library (NaCl). The detailed communication protocols of the HP-VPN gateway are shown in Figure 7.

Three communication entities participate in the whole communication procedures, including a VPN client, HP-VPN gateway and a remote authentication dial in user service (RADIUS) server. The whole communication protocols are mainly composed of five stages, including session initialization, key negotiation, user authentication, network configuration and data transmission. First, nonces and session id are exchanged to initialize a VPN session in the session initialization stage. Second, the public key of VPN client and VPN gateway are sent to peers to generate a session key by Diffie–Hellman algorithm. Then, the client sends its account information to the VPN gateway for remote authentication in the RADIUS server and receives a virtual address if authenticated. Afterward, the client configures the local network by establishing VNIC and adding a system route. Finally, the VPN client starts protected data transmission with the VPN gateway through an encrypted VPN tunnel.

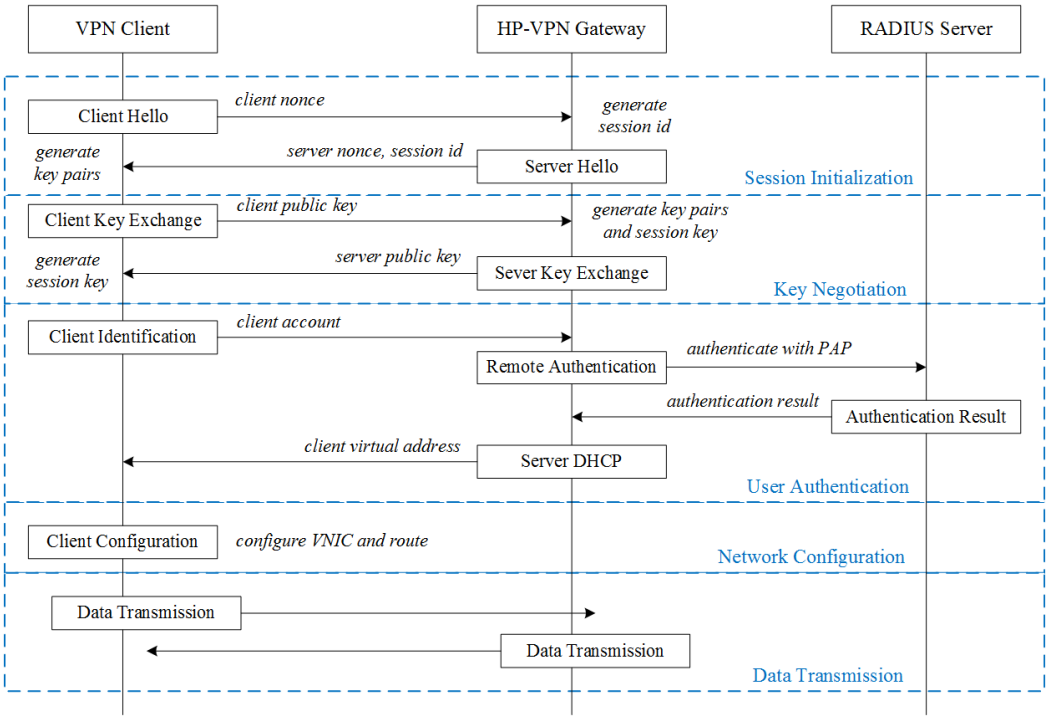

Figure 7. Communication protocols of HP-VPN.

5.2. Traditional VPN (T-VPN) Gateway

A traditional VPN (T-VPN) gateway is implemented using traditional methods, which use a Linux socket to receive and send packets, utilize a kernel protocol stack to decode and encode protocols and apply the combination of VNIC and the kernel Netfilter to implement NAT. In addition, any other hardcores of T-VPN are the same as HP-VPN, including the storage and searching of VPN sessions, the storage and searching of VPN route policies and the VPN protocols as well. The differences of packet receiving and sending, decoding and encoding protocols, and implementation of NAT have been discussed as working mechanisms in Section 3.1.2. Therefore, these details will not be discussed any further.

6. Performance Tests and Evaluations

After implementation of the two prototype systems above, we conduct performance tests on the implemented VPN gateways and other common VPN gateways to prove our proposed GHPA has superior performance. The details are as follows.

6.1. Test Objects

Besides HP-VPN and T-VPN, we chose another four open-source VPN projects: PPTP, Accel-ppp, Strongswan and OpenVPN as our test objects, which basically covers the common VPN protocols, such as PPTP VPN, IPSec VPN and SSL VPN.

6.2. Test Environment

Our performance tests are carried out in a LAN environment. As shown in Figure 8, the topology consists of one gateway device and two PCs. Details of these device configurations are shown in Table 2. In our experiments, the VPN servers of test objects are deployed on the gateway while VPN clients are configured or installed on the PCs.

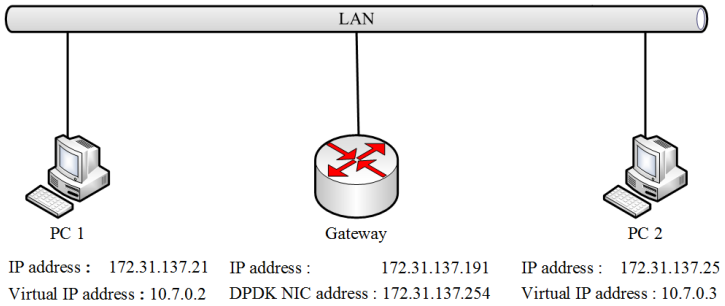

Figure 8. Topology of experimental environment.

Table 2. Details of device configuration parameters.

Names	Parameters	Descriptions
Gateway	CPU	Intel Atom D525 @1.80 GHz
	RAM	1G
	NIC	Intel 82583V Gigabit NIC
	OS	CentOS Linux release 7.2.1511
PC 1/PC 2	CPU	Intel Core i5-4570 CPU @3.20 GHz
	RAM	4G
	NIC	Realtek 8411 PCI-E Gigabit NIC
	OS	Ubuntu 14.04.2 LTS

6.3. Test Process

According to Section 3.1.3, RTT, UDP throughput, UDP packets per second, UDP jitter and TCP throughput are supposed to be tested during the testing process. Meanwhile, considering the two communication modes of VPN gateways, we use Nping and Iperf, two open-source network test tools, to conduct two performance tests for both relay-mode VPN gateways and agent-mode VPN gateways. Each performance test has six rounds for six test objects, respectively. The test process of each round is described as follows.

The test of the relay-mode VPN gateways starts when a VPN server and the corresponding VPN clients are successfully started. First, we execute command ping [address] -c [times] in PC 1 to test RTT and record the test results. Then, we execute command iperf -u -c [address] -l [length] -b [bandwidth] -t [interval] in PC 1, observe and record the maximum UDP throughput, UDP packets per second and UDP jitter of different payload lengths on the premise of a zero packet loss rate. Finally, we test TCP throughput by executing iperf -c [address] -M [mss] -t [interval] in PC 1 and recording the test results.

In addition, the process of testing agent-mode VPN gateways is similar to the process above, so further details will not be stated. The differences between the two test processes are the value of parameter [address]. In the relay-mode tests, the parameter [address] is set to the virtual address of PC 2. In the agent-mode tests, the parameter [address] is set to an address that is not equal to the virtual address of PC 2. Therefore, the corresponding route and DNAT rules must be set on the gateway and PC 2 according to the specific situation.

6.4. Tests Results and Analyses

The test results of the relay-mode VPN gateways and agent-mode VPN gateways are shown in Figures 9 and 10, respectively.

According to the test results of the relay-mode VPN gateways in Figure 9, HP-VPN performs the best in all the evaluating standards mentioned in Section 3.1.3. The details are as follows.

As shown in Figure 9a, the RTT of HP-VPN is obviously lower than that of other VPN gateways. It is further calculated that the average RTTs of the six VPN gateways in ascending order are 0.712 ms, 0.896 ms, 0.978 ms, 1.139 ms, 1.201 ms and 1.599 ms for HP-VPN, Strongswan, T-VPN, Accel-ppp, OpenVPN and PPTP, respectively, which proves HP-VPN provides a better low-latency VPN service than other VPN gateways.

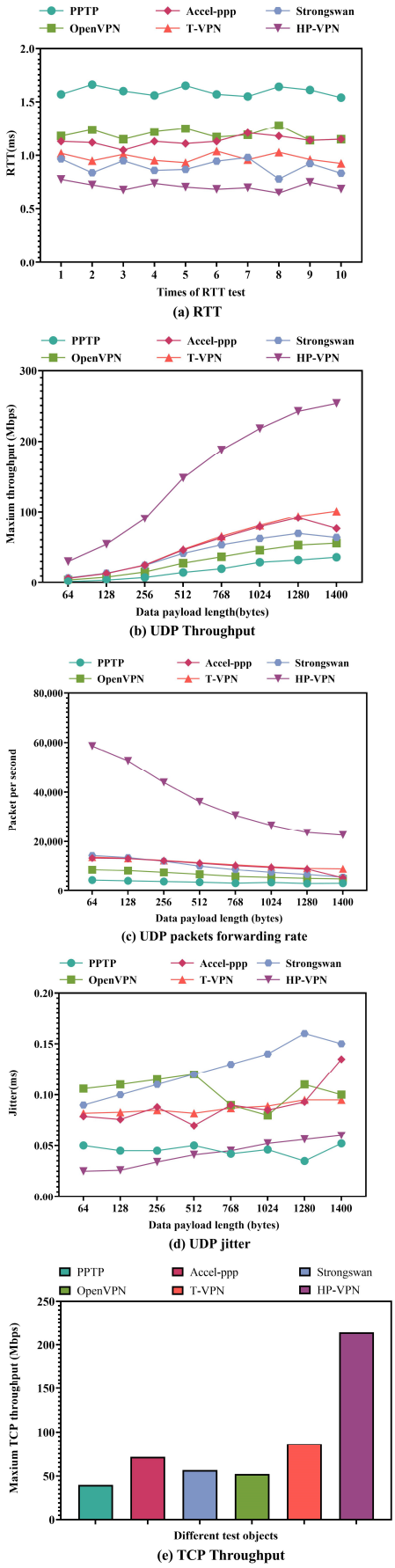

Figure 9. Test results of relay-mode VPN gateways.

Figure 9b depicts the changes in maximum UDP throughput under the circumstances of different data payload lengths. It is clear that the UDP throughput of HP-VPN is far higher than the other five VPN gateways. In view of the overall trends of all curves, maximum UDP throughput rises with increase in data payload length while curve gradient decreases with increase in data payload length. Therefore, a shorter data payload challenges system throughput more. Simply speaking, maximum performance differences among the six VPN gateways can be calculated when testing with the shortest data payload, which will be discussed in Section 6.5.

Figure 9c shows the changes in UDP packet forwarding rate under the conditions of different data payload lengths. The packet forwarding rate of HP-VPN is obviously higher than the other VPN gateways. Meanwhile, the packet forwarding rate decreases with increase in data payload length and the overall trends of all curves are contrary to Figure 9b.

Figure 9d displays the changes in UDP jitter under the circumstances of different data payload lengths. The curves of HP-VPN and PPTP perform relatively few fluctuations than the other VPN gateways. Then, it is calculated that the average jitter of the six VPN gateways in ascending order is 0.042 ms, 0.045 ms, 0.073 ms, 0.087 ms, 0.090 ms and 0.125 ms for HP- VPN, PPTP, OpenVPN, T-VPN, Accel-ppp and Strongswan, respectively, which verifies HP-VPN provides a more stable VPN service than other VPN gateways.

Figure 9e shows the maximum TCP throughput of the six VPN gateways. There is no doubt that the maximum TCP throughput of HP-VPN is far higher than the other five VPN gateways.

Similarly, HP-VPN performs the best in all the evaluating standards in the performance tests of agent-mode VPN gateways as well, ranging from the RTT test results in Figure 10a to the maximum TCP throughput test result in Figure 10e. Considering the process and evaluating standards of testing the two mode VPN gateways are the same, detailed analyses need not be repeated again. However, it is worth mentioning that all VPN gateways show better performance in agent mode than in relay mode. That is because more computational overhead and time overhead are required to encrypt and encapsulate VPN tunnel packets one more time in relay-mode VPN gateways.

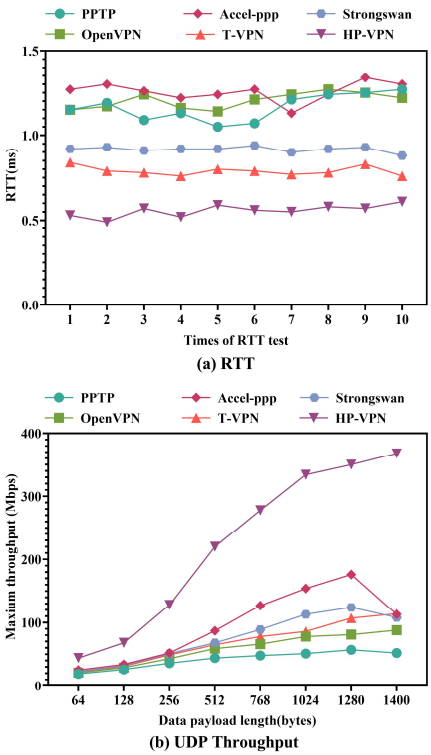

Figure 10. *Cont.*

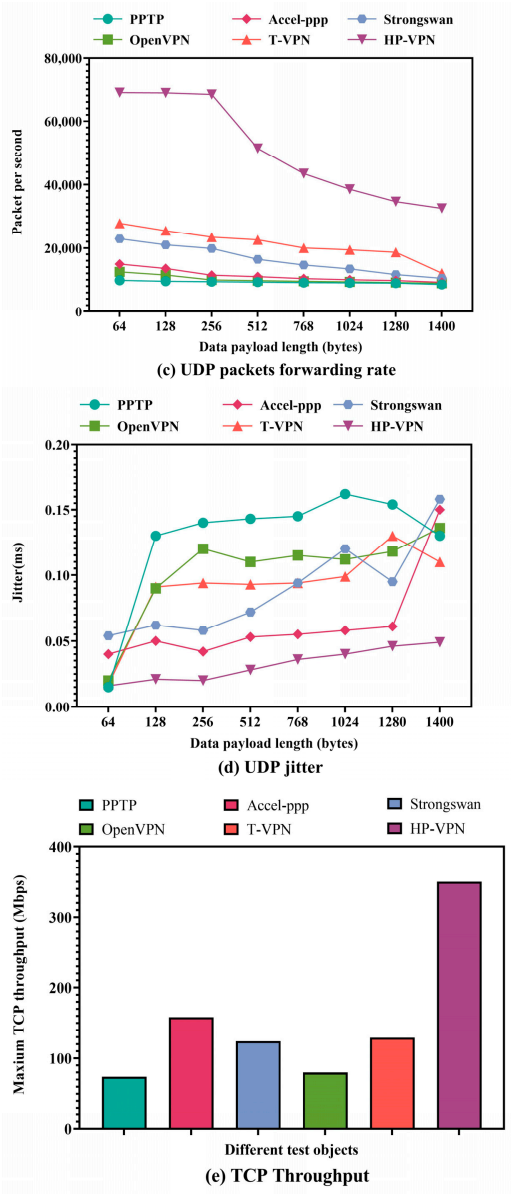

Figure 10. Test results of agent-mode VPN gateways.

Ordinary one-way analysis of variance (ANOVA) is executed from the test results of Figures 9 and 10. The *F* and *p* values for each group are listed in Table 3. The calculation results verify that the differences between the means of each group are statistically significant.

Table 3. Results of ANOVA.

Test Results	*F* Value	*p* Value	Significant Diff. among Means ($p < 0.05$)
Figure 9a	405.20	<0.0001	Yes
Figure 9b	10.20	<0.0001	Yes
Figure 9c	33.00	<0.0001	Yes
Figure 9d	36.14	<0.0001	Yes
Figure 10a	329.00	<0.0001	Yes
Figure 10b	8.978	<0.0001	Yes
Figure 10c	40.06	<0.0001	Yes
Figure 10d	7.197	<0.0001	Yes

6.5. Evaluations

It is observed that the performance of the HP-VPN gateway based on GHPA is superior to the other five VPN gateways in RTT, UDP throughput, UDP packet forwarding rate, UDP jitter and TCP throughput either in relay mode or in agent mode. Considering the PPTP VPN gateway shows the worst performance from the overall test results, we set all evaluating standards of PPTP VPN to index 1 and then calculate the specific performance differences of all test objects both in relay mode and in agent mode, which are shown in Tables 4 and 5, respectively.

Table 4. Performance differences of relay-mode VPN gateways.

Evaluating Standards	PPTP	Accel-ppp	Strongswan	OpenVPN	T-VPN	HP-VPN
Average RTT	1	0.71	0.56	0.75	0.61	0.45
UDP throughout	1	1.67–3.15	1.76–3.21	1.54–1.97	2.77–3.26	7.03–13.04
UDP packet forwarding rate	1	1.67–3.15	1.76–3.21	1.54–1.97	2.77–3.17	7.03–13.04
Average UDP jitter	1	1.96	2.73	1.59	1.91	0.93
TCP throughout	1	1.79	1.42	1.31	2.16	5.35

Table 5. Performance differences of agent-mode VPN gateways.

Evaluating Standards	PPTP	Accel-ppp	Strongswan	OpenVPN	T-VPN	HP-VPN
Average RTT	1	1.08	0.80	1.04	0.68	0.49
UDP throughout	1	2.02–3.70	1.94–2.99	1.26–1.61	1.66–2.00	5.65–9.93
UDP packet forwarding rate	1	2.01–3.71	1.94–2.99	1.26–1.61	1.66–2.10	75.65–9.93
Average UDP jitter	1	0.56	0.77	0.88	0.74	0.28
TCP throughout	1	2.16	1.52	0.90	1.59	4.62

From the perspective of transverse comparisons between HP-VPN based on GHPA and other VPN gateways implemented with traditional methods, the HP-VPN gateway is proved to have the best performance in all evaluating standards both in relay mode and in agent mode. From the perspective of vertical comparisons between HP-VPN and T-VPN with same VPN protocols, using GHPA significantly improves the performance of a VPN gateway as well. When working in relay mode, the average RTT of HP-VPN is 0.73 times that of T-VPN; the UDP throughput of HP-VPN is 2.50 to 4.29 times that of T-VPN; the UDP packet forwarding rate of HP-VPN is 2.50 to 4.29 times that of T-VPN; the average UDP jitter of HP-VPN is 0.49 times that of T-VPN; and the TCP throughput of HP-VPN is 2.47 times that of T-VPN. When working in agent mode, the average RTT of HP-VPN is 0.72 times that of T-VPN; the UDP throughput of HP-VPN is 3.34 to 5.04 times that of T-VPN; the UDP packet forwarding rate of HP-VPN is 3.34 to 5.04 times that of T-VPN; the average UDP jitter of HP-VPN is 0.38 times that of T-VPN; and the TCP throughput of HP-VPN is 2.90 times that of T-VPN. The refactors of user space VPN by utilizing DPDK-based packet processing layer and user space basic protocol stack prove to overcome the performance bottlenecks of VPN gateways mentioned in Section 3.2 significantly. Quantitative analysis will be conducted to research user space VPN implementations in our future work.

In conclusion, GHPA for VPN gateways is superior to traditional methods of implementing VPN gateways in performance and generality. In terms of performance, GHPA provides a high-throughput, low-latency and stable VPN service for VPN gateways. In terms of generality, GHPA is extensible and applicable for other common VPN gateways. GHPA makes it easier to implement high-performance gateways when implementing VPN protocols to meet specific needs.

7. Conclusions

In the high-speed network environments of cloud computing, the performance of VPN gateways is unsatisfactory. This paper presents a generic performance improvement

method for VPN gateways to meet the demands of generality and high performance in a pure software way. From the perspective of generality, we re-design a generic VPN core framework, including modeling a graph-based generic VPN communication model, formulating generic VPN core technologies and designing corresponding core algorithms. From the perspective of high performance, we propose a three-layer generic high-performance architecture (GHPA) on the basis of a proposed generic VPN core framework. This architecture applies DPDK technology, designs user space basic protocol stack and implements user space VPN NAT to break through the performance bottlenecks of packet receiving and sending, the kernel protocol stack and the virtual network interface card, respectively. Then, two prototype systems are implemented referring to the proposed GHPA and traditional methods, respectively. Experimental results verify that GHPA is able to provide significantly better performance than other traditional implementations of VPN gateways. In addition, the proposed GHPA is extensible and applicable to improve performance of other common VPNs. Future work will focus on the optimization of the three core algorithms mentioned in Section 4.3 and improvements in the HP-VPN protocols mentioned in Section 5.1. The optimization of hash algorithms and improvement of routing algorithms will be settled with high priority. The quantitative analysis of the VPN gateway performance bottlenecks mentioned in Section 3.2 will be carried out comprehensively.

Author Contributions: Conceptualization, C.F. and B.W.; methodology, C.F.; software, C.F., R.M. and G.X.; validation, C.F. and R.M.; formal analysis, C.F.; investigation, C.F. and Y.S.; resources, B.W. and Y.Z.; data curation, C.F. and W.W.; writing—original draft preparation, C.F.; writing—review and editing, W.W. and B.W.; visualization, C.F. and R.M.; supervision, C.F. and W.W.; project administration, C.F., R.M. and G.X.; funding acquisition, B.W. and Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (2021YFB2012400), National Natural Science Foundation of China (62272129) and Key Research and Development Program of Shandong Province (No. 2023CXPT065).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

All acronyms mentioned in this paper are shown in Table A1.

Table A1. Acronyms and descriptions in this paper.

Categories	Acronyms	Descriptions
VPN related	VPN	Virtual private network
	PPTP	Point-to-point tunneling protocol
	Accel-ppp	A high-performance VPN server application for linux.
	IPsec	Internet protocol security
	IKEv2	Internet Key Exchange version 2
	Strongswan	An open-source IPsec-based VPN
	SSL	Secure sockets layer
	OpenVPN	An open-Source SSL-based VPN
	SSTP	Secure socket tunneling protocol
	NIC	Network interface card
	VNIC	Virtual network interface card
	NAT	Network address translation
	SNAT	Source network address translation
	DNAT	Destination network address translation
	ARP	Address resolution protocol

Table A1. *Cont.*

Categories	Acronyms	Descriptions
DPDK related	DPDK	Data plane development kit
	EAL	Environment abstraction layer
	IGB	Intel(R) PRO/1000 PCI Express Gigabit Ethernet adapter driver
	UIO	User space input/output
	PMD	Poll mode driver
	Mempool	An allocator of memory of DPDK libraries
	Mbuf	An storage unit of memory of DPDK libraries
	RX/TX	Receiving and transmitting unit of DPDK libraries
Our work related	GHPA	Generic high-performance architecture
	T-VPN	Traditional VPN
	HP-VPN	High-performance VPN
	libsodium	A portable, cross-compliable networking and cryptography library
	RADIUS	Remote authentication dial in user service
	Nping	An open-source tool for packet generation and analysis measurement
	Iperf	A tool for bandwidth measurements on IP networks.
	ANOVA	Analysis of variance

References

1. Nouhas, H.; Belangour, A.; Nassar, M. Cloud and Edge Computing Architectures: A Survey. In Proceedings of the 2023 IEEE 11th Conference on Systems, Process & Control (ICSPC), Malacca, Malaysia, 16 December 2023; pp. 210–215.
2. Muniswamaiah, M.; Agerwala, T.; Tappert, C.C. A Survey on Cloudlets, Mobile Edge, and Fog Computing. In Proceedings of the 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud), Washington, DC, USA, 26–28 June 2021; pp. 139–142.
3. Deng, J.Y.; Deng, J.K.; Liu, P.H.; Wang, H.; Yan, J.; Pan, D.; Liu, J. A Survey on Vehicular Cloud Network Security. *IEEE Access* **2023**, *11*, 136741–136757. [CrossRef]
4. Chavan, J.; Patil, R.; Patil, S.; Gutte, V.; Karande, S. A Survey on Security Threats in Cloud Computing Service Models. In Proceedings of the 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 25–27 May 2022; pp. 574–580.
5. Kavitha, T.; Hemalatha, S.; Saravanan, T.M.; Singh, A.K.; Alam, M.I.; Warshi, S. Survey on Cloud Computing Security and Scheduling. In Proceedings of the 2022 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 25–27 January 2022; pp. 1–4.
6. Santhanamahalingam, S.; Alagarsamy, S.; Subramanian, K. A study of cloud-based VPN establishment using network function virtualization technique. In Proceedings of the 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 20–22 October 2022; pp. 627–631.
7. Osmani, L.; Toor, S.; Komu, M.; Kortelainen, M.J.; Lindén, T.; White, J.; Khan, R.; Eerola, P.; Tarkoma, S. Secure Cloud Connectivity for Scientific Applications. *IEEE Trans. Serv. Comput.* **2018**, *11*, 658–670. [CrossRef]
8. Dayananda, M.S.; Kumar, A. Architecture for Intercloud Services Using IPsec VPN. In Proceedings of the 2012 Second International Conference on Advanced Computing Communication Technologies, Rohtak, India, 7–8 January 2012; pp. 463–467.
9. Fu, C.L.; He, Q.G.; Wang, B.L.; Han, X.X. A Communication Supportable Generic Model for Mobile VPN on Android OS. In Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, Italy, 27–30 June 2016; pp. 1039–1046.
10. Liu, J.; Gao, N.; Tu, C.; Zhang, Y.; Sun, Y. A Pure Hardware Design and Implementation on FPGA of WireGuard-based VPN Gateway. In Proceedings of the 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Rio de Janeiro, Brazil, 24–26 May 2023; pp. 1220–1225.
11. Turan, F.; de Clercq, R.; Maene, P.; Reparaz, O.; Verbauwhede, I. Hardware acceleration of a software-based VPN. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9.
12. Heinemann, C.; Chaduvu, S.S.; Byerly, A.; Uskov, A. OpenCL and CUDA software implementations of encryption/decryption algorithms for IPsec VPNs. In Proceedings of the 2016 IEEE International Conference on Electro Information Technology (EIT), Grand Forks, ND, USA, 19–21 May 2016; pp. 765–770.
13. Pudelko, M.; Emmerich, P.; Gallenmüller, S.; Carle, G. Performance Analysis of VPN Gateways. In Proceedings of the 2020 IFIP Networking Conference (Networking), Paris, France, 22–26 June 2020; pp. 325–333.
14. Intel. DPDK. 2024. Available online: https://www.dpdk.org/ (accessed on 6 May 2024).
15. Raj, J.R.; Srinivasulu, S. Design of IoT based VPN gateway for home network. In Proceedings of the 2022 International Conference on Electronics and Renewable Systems (ICEARS), Tuticorin, India, 16–18 March 2022; pp. 561–564.

16. Elhanashi, A.; Dini, P.; Saponara, S.; Zheng, Q. Integration of Deep Learning into the IoT: A Survey of Techniques and Challenges for Real-World Applications. *Electronics* **2023**, *12*, 4925. [CrossRef]
17. Arashloo, M.T.; Shirshov, P.; Gandhi, R.; Lu, G.; Yuan, L. A scalable VPN gateway for multi-tenant cloud services. *ACM SIGCOMM Comput. Commun. Rev.* **2018**, *48*, 49–55. [CrossRef]
18. Gugueoth, V. LPMLP-Based Framework for Secure IPsec VPN Cloud Gateway with Advanced Network Monitoring and Issue Resolution. In Proceedings of the 2023 IEEE 12th International Conference on Cloud Networking (CloudNet), Hoboken, NJ, USA, 1–3 November 2023; pp. 18–26.
19. Jiang, H.; Wang, Q.; Zhang, G.; Fan, J. Design and implementationg of an IPsec VPN gateway base on OpenWRT. *J. Phys. Conf. Ser.* **2019**, *1176*, 042007. [CrossRef]
20. MZakaria, I.; Norizan, M.N.; Isa, M.M.; Jamols, M.F.; Mustapa, M. Comparative analysis on virtual private network in the internet of things gateways. *Indones. J. Electr. Eng. Comput. Sci.* **2022**, *28*, 488–497.
21. Lawas, J.B.R.; Vivero, A.C.; Sharma, A. Network performance evaluation of VPN protocols (SSTP and IKEv2). In Proceedings of the 2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN), Hyderabad, India, 21–23 July 2016; pp. 1–5.
22. Redzovic, H.; Smiljanic, A.; Savic, B. Performance evaluation of Software Routers with VPN features. In Proceedings of the 2016 24th Telecommunications Forum (TELFOR), Belgrade, Serbia, 22–23 November 2016; pp. 1–4.
23. Ismoyo, D.D.; Wardhani, R.W. Block cipher and stream cipher algorithm performance comparison in a personal VPN gateway. In Proceedings of the 2016 International Seminar on Application for Technology of Information and Communication (ISemantic), Semarang, Indonesia, 5–6 August 2016; pp. 207–210.
24. Kotuliak, I.; Rybar, P.; Truchly, P. Performance comparison of IPsec and TLS based VPN technologies. In Proceedings of the 2011 9th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stara Lesna, Slovakia, 27–28 October 2011; pp. 217–221.
25. Yi, X.; Wang, J.; Duan, J.; Bai, W.; Wu, C.; Xiong, Y.; Han, D. FlowShader: A generalized framework for GPU-accelerated VNF flow processing. In Proceedings of the 2019 IEEE 27th International Conference on Network Protocols (ICNP), Chicago, IL, USA, 8–10 October 2019; pp. 1–12.
26. Raumer, D.; Gallenmüller, S.; Emmerich, P.; Märdian, L.; Carle, G. Efficient Serving of VPN Endpoints on COTS Server Hardware. In Proceedings of the 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, 3–5 October 2016; pp. 164–169. [CrossRef]
27. Wei, X.; Miao, W.; Zeng, Z.; Wang, Y.; Zhao, H.; He, Y.; Wang, Y.; Deng, J. Research on using dynamic thread pool to improve the performance of VPN gateway. In Proceedings of the 2022 7th International Conference on Computer and Communication Systems (ICCCS), Wuhan, China, 22–25 April 2022; pp. 566–570.
28. Li, Z. HPSRouter: A high performance software router based on DPDK. In Proceedings of the 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon, Republic of Korea, 11–14 February 2018; pp. 503–506.
29. Zhang, Y.; Li, Z.; Mei, S.; Xiao, L.; Wang, M. A New Approach for Accelerating IPSec Communication. In Proceedings of the 2009 International Conference on Multimedia Information Networking and Security, Wuhan, China, 18–20 November 2009; Volume 2, pp. 482–485.
30. Wu, W.; Crawford, M.; Bowden, M. The performance analysis of Linux networking–packet receiving. *Comput. Commun.* **2007**, *30*, 1044–1057. [CrossRef]
31. Ntop. PF RING ZC. 2024. Available online: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/ (accessed on 6 May 2024).
32. Rizzo, L. Netmap: A novel framework for fast packet I/O. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 8–10 August 2012; pp. 101–112.
33. Barbette, T.; Soldani, C.; Mathy, L. Fast userspace packet processing. In Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Oakland, CA, USA, 7–8 May 2015; pp. 5–16.
34. Gallenmüller, S.; Emmerich, P.; Wohlfart, F.; Raumer, D.; Carle, G. Comparison of frameworks for high-performance packet IO. In Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Oakland, CA, USA, 7–8 May 2015; pp. 29–38.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.