future internet

MDPI

Article

Overlay and Virtual Private Networks Security Performances Analysis with Open Source Infrastructure Deployment

Antonio Francesco Gentile [1], Davide Macrì [1], Emilio Greco [1,*] and Peppino Fazio [2,3]

[1] Institute for High-Performance Computing and Networking (ICAR), National Research Council of Italy (CNR), Via P. Bucci 8/9C, 87036 Rende, Italy; antoniofrancesco.gentile@icar.cnr.it (A.F.G.); davide.macri@icar.cnr.it (D.M.)
[2] Department of Molecular Sciences and Nanosystems, Ca' Foscari University of Venice, Via Torino 155, 30123 Venezia, Italy; peppino.fazio@unive.it
[3] Department of Telecommunications, VSB—Technical University of Ostrava, 708 00 Ostrava, Czech Republic
* Correspondence: emilio.greco@icar.cnr.it

Abstract: Nowadays, some of the most well-deployed infrastructures are Virtual Private Networks (VPNs) and Overlay Networks (ONs). They consist of hardware and software components designed to build private/secure channels, typically over the Internet. They are currently among the most reliable technologies for achieving this objective. VPNs are well-established and can be patched to address security vulnerabilities, while overlay networks represent the next-generation solution for secure communication. In this paper, for both VPNs and ONs, we analyze some important network performance components (RTT and bandwidth) while varying the type of overlay networks utilized for interconnecting traffic between two or more hosts (in the same data center, in different data centers in the same building, or over the Internet). These networks establish connections between KVM (Kernel-based Virtual Machine) instances rather than the typical Docker/LXC/Podman containers. The first analysis aims to assess network performance as it is, without any overlay channels. Meanwhile, the second establishes various channels without encryption and the final analysis encapsulates overlay traffic via IPsec (Transport mode), where encrypted channels like VTI are not already available for use. A deep set of traffic simulation campaigns shows the obtained performance.

Keywords: IPsec; Linux; OpenWrt; Overlay; LibreSwan; IKE; TLS; Cybersecurity

check for updates

Citation: Gentile, A.F.; Macrì, D.; Greco, E.; Fazio, P. Overlay and Virtual Private Networks Security Performances Analysis with Open Source Infrastructure Deployment. *Future Internet* **2024**, *16*, 283. https://doi.org/10.3390/fi16080283

Academic Editor: Carlo Blundo

Received: 24 June 2024
Revised: 1 August 2024
Accepted: 2 August 2024
Published: 7 August 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

Last-generation IT systems can span vast geographical regions, necessitating secure and dependable infrastructures that ensure cost-effectiveness regarding space and time. Virtual Private Networks (VPNs) and Overlay Networks (ONs) emerge as highly dependable technologies catering to these requirements, seamlessly traversing both traditional Public Switched Telephone Networks (PSTN) and cutting-edge 4G/5G architectures [1–3]. Just for the sake of clarity, we recall that a VPN is based on a technology that establishes an encrypted secure connection over a public network (typically the Internet), enabling users to transmit data securely. VPNs are often utilized to maintain privacy and security during internet browsing, particularly on public Wi-Fi networks, and to circumvent geographic restrictions by hiding the user's IP address and location. An ON, on the other hand, is essentially a virtual network constructed atop an already established physical network infrastructure. Typically, it entails establishing extra logical connections, protocols, or network segments to facilitate particular functions or services, all without necessitating alterations to the foundational network structure. These overlay networks find widespread application in implementing functionalities like Content Delivery Networks (CDNs) and Peer-to-Peer (P2P) networks. They offer adaptability and scalability, enabling the smooth deployment of new services or enhancing existing ones while leaving the underlying network undisturbed. It is important to emphasize that the main differences between an

overlay network and a VPN concern their approach to security, architecture, and managing data flow over a public network. In particular:

- Approach to Security: ONs implement security through end-to-end encryption of data. Each node encrypts the data before transmitting them over the public network. This ensures that the data remain secure even when transmitted over an untrusted network. VPNs, on the other hand, use tunneling to ensure data security. Data are encapsulated in encrypted packets as they traverse the public network. This creates a secure tunnel between devices communicating through the VPN.
- Architecture: ONs are built on top of existing networks, adding an abstraction layer. Nodes in the ON can be implemented on different hardware and software, allowing for greater flexibility. VPNs are typically implemented on existing Internet Protocol (IP) networks, both on the Internet and as part of private networks. They use specific protocols to establish secure connections between participating devices.
- Data Flow Management: ONs manage data flow in a distributed manner. Each node in the network is responsible for transmitting and receiving data, which may follow different paths based on the overlay network topology. VPNs manage data flow in a centralized way. Data are sent through the VPN tunnel between a client and a central VPN server, which handles routing and data security.
- Scalability: ONs can be more scalable than VPNs in some situations, as they can better adapt to changes in network topology and support a larger number of nodes without impacting performance. VPNs may have scalability limitations due to their need to manage centralized VPN tunnels and their dependence on a device's capacity to handle encrypted traffic.

From the sentences above, it is clear that both ONs and VPNs offer solutions for creating private and secure networks over public network infrastructures. Still, their approaches to security, architecture, and data flow management differ. The choice between the two depends on the specific requirements of the network environment and applications.

This work's main aim is to offer a deep performance comparison (from different points of view) between VPNs and ONs. The scenario in which we operate relates to the analysis of a network's performance (RTT and bandwidth), since the type of overlay network used for interconnecting traffic between two VMware Virtual Machines varies (in the future, hardware-constrained devices such as Raspberry Pi 4/5 will be used). The networks connect KVM (Kernel-based Virtual Machine) machines instead of Docker/LXC/Podman containers. We only use two network namespaces directly in the *MACsec* case. The former resides on the first physical endpoint, while the latter resides on the other. The first guest (virtual machine) resides on the first physical host, and naturally, the second guest resides on the same physical host. The host machine has the following specifications: 12th Gen Intel(R) Core(TM) i7-1280P 2.00 GHz processor family, RAM 32GB LPDDR5-SDRAM, 4800 MHz, and 1 TB SSD (ASUSTeK Computer Inc. Taipei, Taiwan). One of the main results we want to achieve with this work is to group the tunnels according to the Achievable RTT, therefore having a wide variety of choices that respect a certain RTT distribution. We will also demonstrate the behavior of UDP compared to TCP, due to channel saturation. In this work, different ON tunnels are considered: *GRE*, *GREtap*, *FOU*, and *GUE*, *IPIP*, *GENEVE*, *VXLAN*, and *MACVLAN*. Also, different IPsec approaches are considered: *IKEv1/IKEv2 Tunnel Mode*, *IKEv1/IKEv2 Transport Mode*, *IKEv2 VTI ROUTE BASED*, and *IKEv2 XFRMi ROUTE BASED*. We also consider different TLS/NOISE-based VPNs: *OpenVPN*, *OpenConnecs Ocserv*, *Wireguard*, and *Tinc VPN*. Finally, as a pure overlay with native encryption tunnels, we consider *Nebula Overlay* and *MACsec*. For the sake of completeness, it is reported that these networks can also be configured to create *Site to Site*, *ROAD WARRIORS*, *MESH*, and *HUB & SPOKE* topologies. Of course, not all of them have all these features, but they are designed for specific purposes. This paper is organized as follows: Section 2 is dedicated to related works on VPN realizations and evaluates the existing literature. Section 3 gives a deep overview of a VPN's architecture. Section 4

describes the scenarios used for our testbeds, while Section 5 presents the experimental results. Finally, Section 6 summarizes the paper.

2. Related Works

Numerous studies in the literature make suggestions regarding security systems across various protocol stack layers and network typologies, including cryptography performance analysis; this section provides an overview of key contributions related to VPNs and ONs.

In [4], a novel experimental WAN solution as Software-Defined Wide Area Network (SD-WAN based on ONOSOpenDaylight https://opennetworking.org/onos/ (accessed on 23 June 2024) and OpenvSwitch 2.12.0 https://www.openvswitch.org/ (accessed on 23 June 2024) latest version as SDN controller) is proposed, entirely based on secure tunnels, implemented by the Generic Routing Encapsulation (GRE) tunneling protocol [5]. The proposal's effectiveness is validated through two SD-WAN testbeds, covering a very long distance (from northeast to south Italy). The authors demonstrate that the SD-WAN can ensure a rapid recovery from and resilience to network failures by leveraging an innovative Berkeley Packet Filtering (BPF)-based monitoring technique. The authors also demonstrate that SD-WAN has several advantages in terms of recovery time and high performance compared to other solutions (e.g., Multi-Protocol Label Switching—MPLS).

The study in [6] evaluates network layer-based VPNs, which enable secure communications between remote LANs using IP tunnels over a shared medium (e.g., Internet). The authors' primary focus lies in demonstrating the efficiency of various VPNs, backed by several studies on contemporary practices, followed by an analysis of the Wireguard protocol. The authors of [7] provide a comprehensive analysis of MACVLAN and IPvlan modes, focusing on their architecture characteristics and utility in enabling container communication across diverse network segments within complex application scenarios. The proposed work details the setup of container overlay networks in single-machine and multi-machine environments, offering insights into the intricacies of these modes. By evaluating key network indicators such as round-trip time (RTT), bandwidth, and jitter using tools like Ping and Iperf, the research assesses the end-to-end communication performance of containers. The experimental results indicate that the IPvlan mode demonstrates superior network performance compared to MACVLAN, presenting a more extensive and adaptable application scenario for container overlay networks. This finding highlights the significance of selecting the appropriate network mode based on specific performance and scalability requirements within containerized environments. The detailed contribution given in [8] presents a comprehensive approach to studying VPNs in general. It emphasizes that understanding VPN technology necessitates consideration of the values and motivations of individuals within organizations. A significant discovery highlights the divergent perceptions and interpretations of the terms *VPN*, *security*, and *privacy* among corporate employees. In [9] the attention id focuses on the widespread utilization of Secure Socket Layer (SSL)-VPN to ensure both authentication and data confidentiality between the client (typically a web browser) and the server (SSL Server). VPN technology is extensively examined in [10,11]. Specifically, these works delve into the vulnerabilities and suboptimal performance experienced under heavy loads by web browser-based SSL VPNs, which are limited to support only Windows operating systems. The authors of [12] justify the selection of the IPsec-IKEv2 suite over other potential implementations, such as SSL/TLS VPN or SSH Tunnel, by emphasizing the security of the IP communication layer. This choice is motivated by the desire for *transparency* of the layers above the IP within IPsec-IKEv2, as well as the ability to promptly update encryption in case of a compromise in the data channel. The research in [13] examines the use of IPsec for establishing VPNs tailored to e-business requirements. It utilizes a Linksys router as an access point and a VPN concentrator, with OpenSWAN as the system daemon. The setup employs IKEv1-L2TP for authentication and traffic management and IPsec for encryption. The article [14] proposes an approach to deploy VPN technologies for securely encrypting traffic in untrusted networks. This includes environments like bars, lounges, conferences, free hotspots, airport Wi-Fi, and,

more broadly, during any travel scenario. The IPsec stack generally consists of three main parts: Encapsulating Security Payload (ESP), Authentication Header (AH), and the Internet Key Exchange protocol (IKE). The previously mentioned work [13] discusses the AH protocol with an emphasis on VPN security and Quality of Service (QoS) aspects. The IKE protocol can generate session keys for secure communications between VPN endpoints through a proper message exchange, with two existing versions: IKEv1 and IKEv2 [9,15,16]. IKEv1 offers extensive configuration options but suffers from architectural complexity, making it difficult to deploy in modern networks. In contrast, IKEv2 improves upon IKEv1 by addressing its limitations and providing more efficient encryption, with native support on platforms like OS X 10.11+, iOS 9.1+, Linux 3.x+, and Windows 8+. Additionally, IKEv2 is well-supported on mobile devices and compatible with IKEv2-aware clients and third-party iOS, Android, Blackberry, and Windows applications. In [17], a comparative analysis of VPN technologies such as L2TP, PPTP, OpenVPN, Ethernet over IP (EoIP), and MPLS evaluates their suitability for specific business needs based on performance metrics and packet transit characteristics. In [11], the focus is on the authentication parameters required for VPN server access. While some implementations may only necessitate the traditional username/password combination, other deployments like IKEv1-XAUTH may require additional parameters. Microsoft's Secure Socket Tunneling Protocol (SSTP) [18,19] is proposed to channel PPP device traffic through an encrypted VPN tunnel via HTTP using SSL/TLS transport, handling key negotiation, channel encryption, and traffic integrity. Quick UDP Internet Connections (QUIC) is highlighted as suitable for IoT devices with limited resources [20] with implementation guidance in general TrAnsPort services (TAPS) [10], as discussed in [12]. VPNaaS simplifies network security by integrating configurations into a single product to meet various corporate security needs and enable access to virtualized networks on the cloud. The implementation of VPNaaS using WireGuard with a 5G WAN backbone is proposed in [21]. In [22], an experimental analysis on a Debian Linux environment implements the IPsec tunneling protocol with various encryption algorithms, concluding that IPsec AES-sha1 performs comparably to IPsec 3DES-sha1. However, UDP VPN encryption/decryption can degrade performance due to high memory/CPU usage.

3. Components of VPN/Overlay Architecture

This section covers the two most popular VPN protocols (IPsec and SSL/TLS-based VPN) and Linux virtual interfaces, specifically examining the following tunnels:

- *IPIP*: The IPIP tunnel is a method of tunneling that allows IP packets to be transmitted within another IP packet. This type of tunnel is primarily used to connect two LANs via the Internet. Due to its minimal overhead, it is optimal for this application, although it only supports IPv4 unicast traffic and not multicast.
- *VTI*: The Virtual Tunnel Interface (VTI) on Linux facilitates IP encapsulations and is compatible with XFRMi. It establishes secure tunnels and enables kernel routing atop. VTI tunnels function similarly to IPIP or SIT tunnels, with the addition of fwmark (https://www.linux.org/docs/man8/tc-fw.html accessed on 23 June 2024) and IPsec encapsulation/decapsulation capabilities.
- *GRE and GREtap*: Generic Routing Encapsulation (GRE), RFC 2784, involves inserting a GRE header between the inner and outer IP headers. Unlike IPIP, which is limited to encapsulating IP, GRE theoretically supports encapsulating any Layer 3 protocol with a valid Ethernet type. GRE tunnels can transport multicast traffic and IPv6. While GRE operates at OSI Layer 3, GREtap works at Layer 2 (with an encapsulated Ethernet header).
- *FOU*: Tunneling operates across different layers of the networking stack, with IPIP or GRE working at the IP level and FOU (Foo Over UDP) functioning at the UDP level. The leverage of UDP tunneling offers several advantages, especially in existing hardware infrastructure, such as RSS in NICs, ECMP in switches, and checksum offload. Performance boosts have been evidenced for IPIP protocols through developer patch sets. Currently, the FOU tunnel accommodates encapsulation pro-

tocols such as IPIP and GRE, with an example FOU header provided. Configuring a FOU receive port for IPIP entails setting it to port 5555, while for GRE, setting ipproto (https://www.ee.torontomu.ca/~courses/ee8205/Data-Sheets/Tornado-VxWorks/vxworks/ref/ipProto.html accessed on 23 June 2024) 47 is required. Another command establishes a new IPIP virtual interface (FOU1) configured for FOU encapsulation, with the destination port set to 5555 .

- *GUE*: Generic UDP Encapsulation (GUE) differs from FOU by including its encapsulation header with the protocol information and additional data, thus supporting inner IPIP and GRE encapsulation, and configuring a GUE receive port for IPIP on port 5555 involves setting up an IPIP tunnel for GUE encapsulation.
- *GENEVE*: Generic Network Virtualization Encapsulation (GENEVE) consolidates the functionalities of VXLAN, NVGRE, and STT, aiming to address their perceived limitations. Many researchers anticipate that GENEVE could ultimately supplant these earlier formats entirely. The GENEVE tunnel header closely resembles VXLAN, with the key distinction lying in its flexibility. The GENEVE header allows for easy integration of new features through extension with a new Type–Length–Value (TLV) field.
- VXLAN: VXLAN (Virtual eXtensible Local Area Network) is a tunneling protocol that overcomes the limitations of VLAN IDs (4096) by introducing a 24-bit VXLAN Network Identifier (VNI), enabling up to 2^{24} 16,777,216 virtual LANs as described in IETF RFC 7348. This protocol is widely used in data centers to interconnect virtualized hosts across multiple racks, encapsulating Layer 2 frames with a VXLAN header into UDP-IP packets.
- *IP/MACVLAN*: MACVLAN enables multiple MAC and IP addresses on a single physical interface through MACVLAN sub-interfaces. This is different from VLANs, where sub-interfaces share the same MAC address. Each MACVLAN sub-interface has a unique MAC and IP address directly integrated into the underlay network. Typically employed in virtualization, MACVLAN interfaces allow Containers or VMs to obtain DHCP addresses directly, easing integration into existing networks. MACVLAN offers four types, with the MACVLAN bridge being the most common, enabling local communication without external routing. External connectivity utilizes the underlay network, as illustrated by two Containers communicating via the MACVLAN bridge.
- *MACsec*: MACsec operates at Layer 2, ensuring transparent protection (integrity and/or encryption) within the network. Unlike IPsec, which can pose performance challenges, MACsec is designed to run at line rate, typically in a hardware's ASIC, although it is not universally supported across hardware. The protected MACsec frame utilizes an Ethertype of 0x88e5. IPVLAN is akin to MACVLAN but with a key distinction: the endpoints share the same MAC address. Supporting both L2 and L3 modes, IPvlan offers flexibility in networking configurations. In L2 mode, each endpoint retains the same MAC address but receives a different IP address. Conversely, L3 mode facilitates packet routing between endpoints, enhancing scalability. While the Ethernet Header and SecTag are sent in plaintext, they are always integrity-protected by ICV. The default cryptographic algorithm is AES-GCM-128. Additionally, MACsec supports optional replay protection with a configurable replay window.

Figure 1 illustrates the generic concept of stacking and encapsulation, fundamental to understanding the functioning of Overlay networks and VPNs.

These solutions, backed by commercial products like Mikrotik and open-source ones like OpenWrt and Linux distributions, are both at the forefront of the field and serve as a foundation for new research, encompassing VPN protocols, Linux operating systems, and both proprietary and open-source hardware platforms, as referenced in [23].

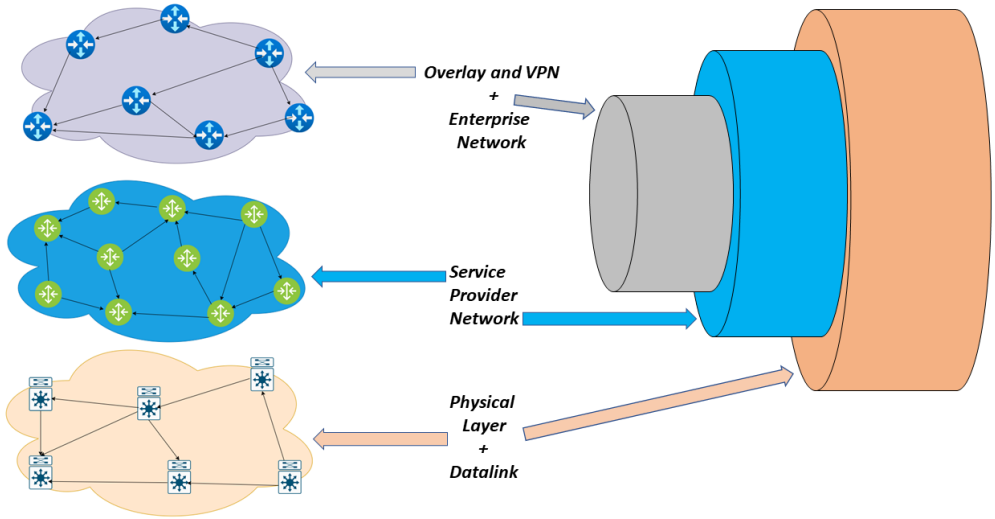

Figure 1. Generic VPN/Overlay stacking technology.

3.1. IPsec

IPsec, or Internet Protocol Security, manages encrypted Virtual Private Network (VPN) tunnels at the OSI Layer 3 level. It forms part of a suite of protocols standardized by the "Internet Engineering Task Force", the organization responsible for advancing Internet technologies. Originally developed for IPv6, IPsec has since been adapted for use with IPv4. It offers three primary functional implementations, as described in [23]. IPsec utilizes AH and ESP protocols to ensure the integrity and authenticity of transmitted data. The AH protocol authenticates the data source and safeguards against packet modification during transmission through the Packet Accelerator Extension. Additionally, it includes a sequence number in the header to prevent packet replay attacks. Meanwhile, the ESP protocol verifies data integrity and identity and encrypts the payload data. However, ESP authentication excludes the outermost IP header, necessitating additional encapsulation for correct delivery, particularly across networks employing Network Address Translation (NAT), such as private xDSL networks. Our choice for this article was to use Libreswan implementation on linux systems, as shown in Table 1. IPsec operates in two modes: tunnel and transport. The IP packet header remains unchanged in transport mode, with the transport protocol inserted between the header and the data area. This mode provides security between the source and destination addresses and is suitable for *Host-to-Host* or *Host-to-Router* connections. Conversely, tunnel mode adds a new complete IP header to the data packets, concealing original addresses and data. This encapsulated packet is forwarded between encrypted endpoints defined by the new outermost IP header. Tunnel mode is commonly used for *Site-to-Site*, *Host-to-Site*, and *Host-to-Host* data transmission scenarios.

3.2. TLS Based VPN

TLS VPN, also known as Secure Sockets Layer VPN [8,24,25], offers a standard VPN solution that operates within the Transport Layer and can be accessed through a web browser or specific client/server applications. Communication between the sender and receiver takes place through socket connections. Two common SSL/TLS VPN deployments are referenced in [23]. In traditional SSL/TLS communication, two keys are utilized for data encryption: a public key shared among all users and a private key unique to each endpoint. To enhance security, two-factor authentication or one-time passwords (OTP) may be employed. Unlike IPsec VPNs, where authenticated clients have full access to the private network, SSL/TLS VPNs offer more granular control over access. This allows for creating specific tunnels for applications via sockets, rather than granting access to the

entire network, and enables the implementation of specific *access roles* with distinct rights for different users.

Table 1. LibreSwan features table.

Feature	LibreSwan
Pre-shared key authentication	Yes
Public-key authentication	Yes
IKEv1 key exchange	Yes
IKEv2 key exchange	Yes
AH support	Yes
NSS compatibility	Yes
DnsSec/XAUTH	Yes
Network Manager compatibility	Yes
VIP (Virtual IP Pools)	Yes
NAT Traversal	Yes
MOBIKE	Yes
Route-based VPN	Yes
Policy-based VPN	Yes
Native {Policy/Route}–based VPN	Yes
HA (High Availability)	Yes
Legacy cipher suites backwards compatibility	No

3.3. The Noise Protocol Framework

The Noise Framework is a lightweight and flexible cryptographic library designed to provide a wide range of cryptographic primitives for secure communication over unreliable networks, such as the Internet. Used within Nebula Overlay, the Noise Framework offers a robust infrastructure to ensure the security and privacy of communications between network nodes. What follows is a technical overview of how the Noise Framework operates within Nebula Overlay. Supported cryptographic primitives: The Noise Framework offers a set of cryptographic primitives, including symmetric encryption, Diffie–Hellman key exchange, key authentication, digital signatures, and cryptographic hashes. These primitives can be flexibly combined to meet the communication's specific security requirements. Noise protocol patterns: The Noise Framework uses protocol patterns to define the phases and operations involved in secure communication. Protocol patterns, such as the *Noise IK* pattern, specify a sequence of cryptographic actions during message exchange between parties. These protocol patterns provide a standardized framework for designing secure and scalable cryptographic protocols. Selection of protocol pattern: In the context of Nebula Overlay, the Noise Framework allows for the selection of the most suitable protocol pattern based on the application's specific requirements and network configuration. For example, the *Noise IKpsk2* pattern may be used for pre-shared key exchange with authentication, while the *Noise NN* pattern may be suitable for anonymous communication. Configuration of cryptographic parameters: The Noise Framework enables the configuration of cryptographic parameters, such as the choice of encryption algorithm, the use of digital signatures, and the setting of Diffie–Hellman parameters. This configurability allows for adjusting the security level and communication performance based on the specific requirements of the application and network environment. Integration with Nebula Overlay: The Noise Framework establishes secure and authenticated communication channels between network nodes within Nebula Overlay. Noise-based communication protocols enable nodes to exchange data securely, ensuring the transmitted information's confidentiality, integrity, and authenticity. In summary, the Noise Framework is a fundamental component within Nebula Overlay for ensuring communication security over unreliable networks. By using flexible protocol patterns and a wide range of cryptographic primitives, the Noise Framework provides a scalable infrastructure for distributed communication over the Internet.

3.4. Operative Systems Hardware Platforms and Linux Daemons

Currently, numerous operating systems and HW/SW platforms are tailored for networking and secure communications. For example, LibreSwan [26] and StrongSwan [27] allow Linux and BSD systems to implement the IPsec protocol. These are referenced in [23]. Table 1 below provides a list of features for the LibreSwan suite. In this article, we will compare the OpenWrt and Debian Native VMs to realize the testbeds.

OpenWRT

Initially designed for wireless routers, OpenWRT [28] is a distribution aimed at enhancing the functionality of devices beyond what is provided by manufacturer-supplied firmware. This operating system ensures a filesystem with user-writable permissions, enabling users to install third-party software and extend the device's capabilities. By utilizing the latest routing software, OpenWRT offers enhanced security and fewer software bugs compared to pre-installed manufacturer software, particularly on older devices that are no longer supported. Additionally, OpenWRT can be installed on custom hardware, including specific boards. For x86-64 platforms, it supports virtualization, allowing the creation of small networks of containers for on-demand services and facilitating network access for all LAN-connected devices.

4. Description of Possible Deployment for Overlay Topologies

To deploy a VPN, many HW and SW solutions are available (e.g., use Mikrotik RouterBoards, *Mikrotik*, https://mikrotik.com/ accessed on 23 June 2024, or, for example, with a Debian OS *DEBIAN OS* https://www.debian.org/index.it.html accessed on 23 June 2024).

Three macro application scenarios, shown in Figures 2–5, are described in [23] and deployed both with VPNs and Overlay TLS.

These solutions guarantee secure transit through encryption algorithms such as AES (128–512) and elliptic curve implementations like ecp192. They offer a range of authentication methods, from PSK + XAUTH to certificate-based and two-step solutions using OTP or additional credential files like RADIUS + VPN IPsec/OpenVPN/OpenConnect, tailored for diverse devices and networks.

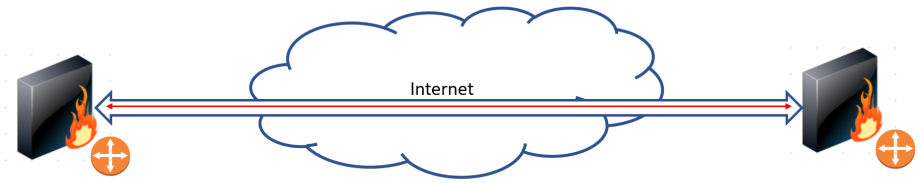

Figure 2. Site-to-site Overlay network scenario.

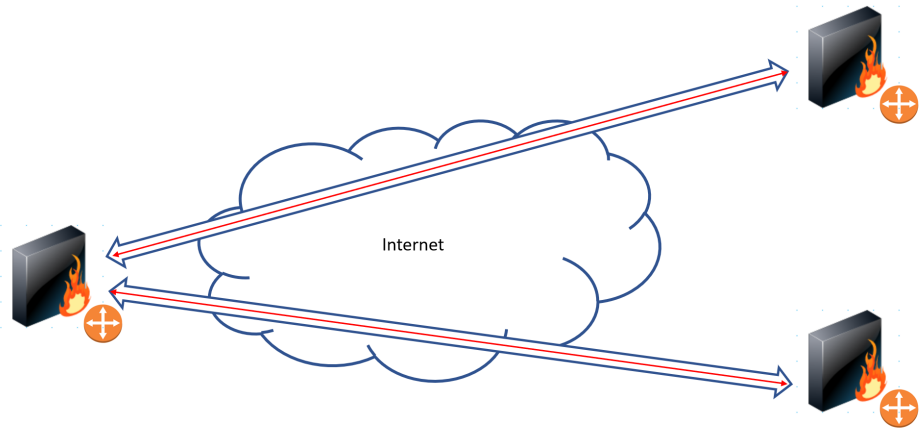

Figure 3. Site-to-multi-site Overlay network scenario.

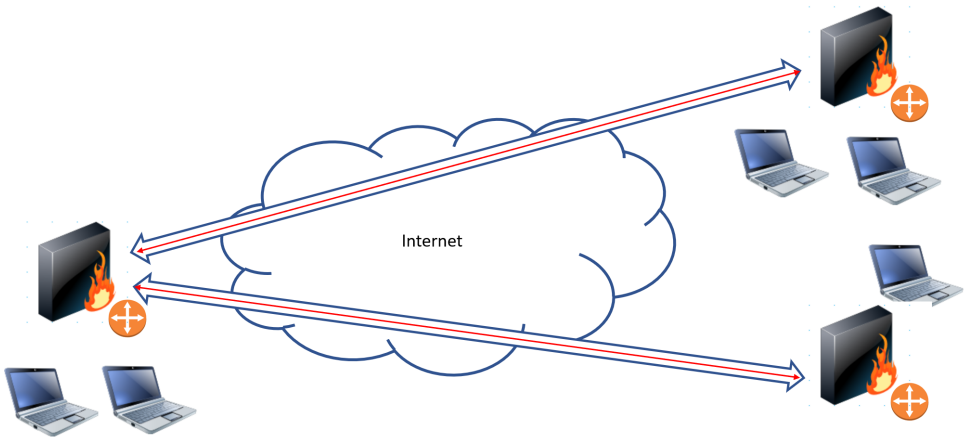

Figure 4. Hub and spoke Overlay network scenario.

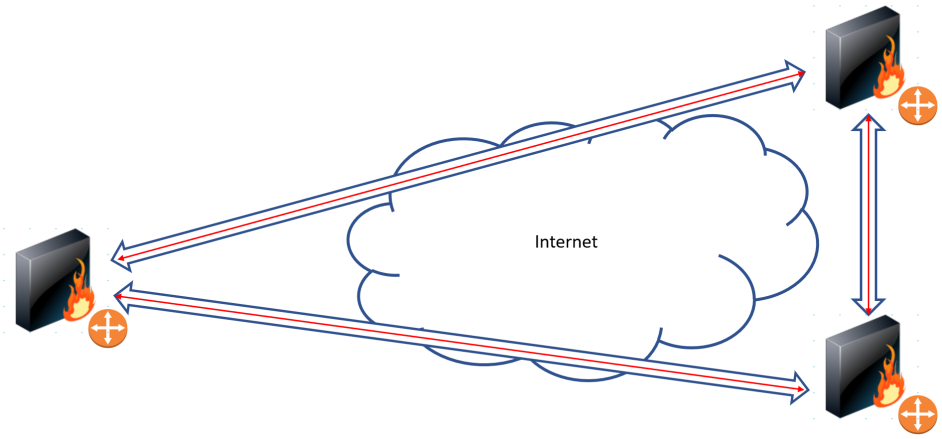

Figure 5. Mesh Overlay network scenario.

This work assesses the current landscape of communication infrastructures (WI-FI, CABLE, 4G/5G) and the devices that can be connected with those infrastructures (NUC, mini PC, Raspberry Pi, Smartphones). It aims to provide the broadest support for various devices and technologies while ensuring optimal performance and security.

Table 2 summarizes the implemented testbeds, outlining the network infrastructure and software used. Several of the proposed architectures remain in active use, including OpenVPN and WireGuard, as part of the COGITO project, https://www.icar.cnr.it/progetti/cogito-sistema-dinamico-e-cognitivo-per-consentire-agli-edifici-di-apprendere-ed-adattarsi/ accessed on 23 June 2024, Themes to interconnect the cloud and local offices.

With the rapid expansion of technologies such as IoT [29] and the evolution of communication paradigms like Fog Computing and cloud-based solutions (e.g., AMAZON AWS), or multi-site clusters (e.g., through Kubernetes), it is essential to manage the increasing data volume while ensuring compliance with GDPR.

The OpenWRT operating system allows these technologies to be implemented on compatible routers and devices. The wide range of supported bridges, particularly for IoT networks, enables interfacing with third-party protocols (e.g., LoRa and ZigBee) and traditional TCP/IP stack protocols. The x86-64 versions support virtualization, allowing for the secure and cost-effective creation of Fog networks of microservices and IoT multi-protocol networks. For instance, outdoor sensor networks can be installed in areas covered by 4G/5G networks and configured locally using an Edge Computing approach.

Table 2. A summary of the VPN-implemented testbeds.

VPN SOFTWARE MANAGED	VPN DEPLOY	VPN IMPLEMENTERS	OPERATIVE SYSTEMS	PLATFORMS
IPsec LibreSwan 4.15	Site to Site	IKEv2 PSK TUNNEL IKEv2 PSK TRANSPORT	Linux DEBIAN 12 WINDOWS 10/11 (client) DEBIAN 11/12 (client) ANDROID 11 (client) iOS 16 (client) MAC OS X 14 (client) RASPBERRY Pi 2/3/4 OpenWRT 23.x	armv7 x86 x86-64 ARM64 ARM MIPSBE MMIPS SMIPS PPC
IPsec LibreSwan 4.15	Site to Site	IKEv2 XFRMi/VTI ROUTE BASED	Same as above	Same as above
OpenVPN 2.6.3	Site to Site		Same as above	armv7 x86 x86-64
Wireguard 1.0.2	Site to Site		Same as above	armv7 x86 x86-64 ARM64 ARM MIPSBE MMIPS SMIPS PPC
Ocserv 1.1.6	Site to Site		Same as above	armv7 x86 x86-64
Tinc VPN 1.0.3.6	Site to Site		Same as above	armv7 x86 x86-64 ARM64 ARM MIPSBE MMIPS SMIPS PPC
Nebula Overlay VPN 1.6.1	Site to Site Host to Host		Same as above	armv7 x86 x86-64 ARM64 ARM MIPSBE MMIPS SMIPS PPC

5. Implemented Scenarios and Experimental Results

To assess the efficiency of various VPN protocols across different OS and HW configurations illustrated in Table 2, various network configurations were established as depicted in Figures 2–4. Using guest KVM stems from the ability to nest container networks, possibly using a *MACVLAN* approach instead of a classical bridge, directly on individual guest networks. This would enable harnessing the capabilities of both virtual machines, such as resilience, and containers, such as lightweightness and portability.

The tools used to capture network traffic data during various VPN and Overlay Tunnel sessions are those referenced in [23] and the network hosts topology is shown in Table 3.

Table 3. Network topologies HW components.

Hardware	Quantity
Workstation with 12th Gen Intel(R) Core(TM) i7-1280P (Santa Clara, CA, USA)—2.00 GHz, 32GB RAM	1
VMWare VM Debian 12 x86-64 virtualized	2
VMWare VM Alpine Linux Latest x86-64 virtualized	1
VMWare VM OpenWRT 23.x x86-64 virtualized	1

Listings 1–3 (bash scripts) depict the script executed by Ansible to activate a tunnel (in this case a FOU type) and automatically configure the network settings and kernel firewall parameters.

Listing 1. Bash script for a FOU interface.

```
# MAIN - left-fou-iface-up.sh

#!/bin/bash

IFACE_MTU="1472"
LOCAL_KVM_NET="192.168.122.0/24"
REMOTE_KVM_NET="192.168.123.0/24"

echo "Setting up FOU TUNNEL left side ..."
modprobe ipip
ip fou add port 5555 ipproto 4
ip link add name fou0 type ipip remote ${HOSTB} local ${HOSTA} mode ipip
 ↪  ttl 255 dev eth0 encap for encap-sport 5555 encap-dport 6666
ip link set dev fou0 up mtu ${IFACE_MTU}
ip a a 10.0.0.1 peer 10.0.0.2 dev fou0
ip r a ${REMOTE_KVM_NET} dev fou0
```

Listing 2. Bash script for a FOU interface iptables rule.

```
# ----- IPTABLES VERSION -----

iptables -t mangle -A FORWARD -o eth0 \
  -p tcp -m tcp --tcp-flags SYN,RST SYN -s ${LOCAL_KVM_NET} \
  -m tcpmss --mss ${IFACE_MTU}:9100 -j TCPMSS --set-mss ${IFACE_MTU}
```

Listing 3. Bash script for a FOU interface Nftables rule.

```
# ----- NFTABLES VERSION -----
# filter
table ip filter {
        chain input {
                type filter hook input priority 0; policy accept;
        }
        chain output {
                type filter hook output priority 0; policy accept;
                counter comment "count accepted packets"
        }
        chain forward {
                type filter hook forward priority 0; policy accept;
                ip saddr ${LOCAL_KVM_NET} oifname eth0 tcp flags syn tcp
                 ↪  option max seg size set 1360:9100
                counter comment "count dropped packets"
        }
}
```

The nodes are connected to a gigabit VMWARE Ethernet switch with 1 Gbps links. The complete topology deploys a subnet representing the WAN (with *public IP* assigned) and two private subnets (the KVM networks) for each workstation, as shown in Figure 6. In the *Site-to-Site* scenario, we have two VPN/Overlay Endpoints used as terminations of the tunnels and linked to local hosts that create traffic flows. In the *Site-to-Multi-Site* topology, three endpoints (the number of nodes can be increased) are linked to local hosts through which traffic flows. To generate the network traffic, the *Iperf, Netperf* and ping

tools are used to measure productivity, jitter, bandwidth, and round-trip time (RTT) while several counters of the same operating system measure the CPU usage.

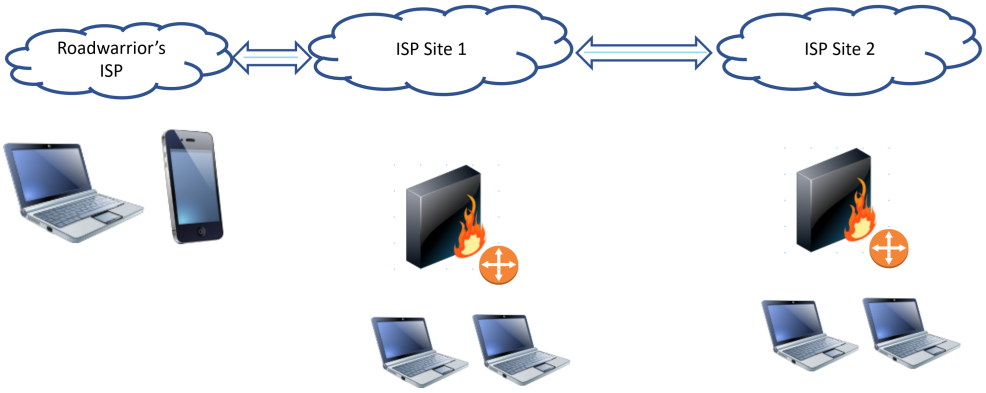

Figure 6. Generic VPN deploy topology.

To run the emulations necessary to collect the data, Ansible (*Ansible Project*, https://www.ansible.com/ accessed on 23 June 2024) was used on the network endpoint machines. An Ansible playbook to manage network tunnels on Linux works by defining a series of steps to be executed on one or more Linux systems to configure the desired network tunnels. These steps typically include:

- Installing necessary packages to support network tunnels, such as OpenVPN, IPSec, or using the scripts to deploy needed configurations.
- Configuring the specific tunnel configuration files, such as OpenVPN ('*.conf') or LibreSwan ('*.conf') configuration files.
- Enabling and starting the necessary services to manage the network tunnels.
- Configuring firewall rules, if necessary, to allow traffic through the network tunnels.
- Verifying the status of network tunnels to ensure they are active and functioning correctly.

The Ansible playbook defines the actions to configure and manage network tunnels on the target Linux system. It allows administrators to automate this process through a series of declaratively defined steps in the YAML playbook file.

5.1. Considered VPN/Overlay Deployed Topology

This paper describes deployments utilized in the *COGITO* (*COGITO project*, https://www.icar.cnr.it/progetti/cogito-sistema-dinamico-e-cognitivo-per-consentire-agli-edifici-di-apprendere-ed-adattarsi/ accessed on 23 June 2024) project, adaptable to any environment, employing selected hardware to measure component performance and ensure backward compatibility. Figure 6 illustrates the basic network topology with example IP address ranges. In Site-to-Site scenarios, the right endpoint is periodically swapped, establishing connections as depicted. Wi-Fi-only devices acting as VPN clients connect via the left endpoint, initially linked to a dummy WAN mimicking a public network (ISP sites 1, 2, and 3) with a router (IP 192.168.21.254) on the 192.168.21.0/24 segment, facilitating communication between VPN endpoints (wired/wireless clients connected to the router or switch).

Ansible facilitated deployment scenarios administration, while throughput results were obtained using *Iperf2* (version 2.2, included in Debian 12 Repository) and *Netperf* (version 2.7, included in Debian 12 Repository), and RTT data using *ping*. Graphics were generated based on the output results from Iperf and Netperf.

5.2. Throughput Performance Analysis

Iperf, available in both Iperf2 and Iperf3 versions as open-source software, is utilized for bandwidth measurement across IP networks. Its compatibility spans major operating systems, including Android and iOS, and its configurable parameters make

it essential for analyzing network performance across diverse protocols, establishing its prominence in network diagnostics. Netperf supports various performance tests, including TCP and UDP, single-stream and multi-stream, burst, and latency tests. It is a flexible and widely used tool in the industry for evaluating network performance and identifying performance issues.

Figure 7 shows the trend of the average RTT for different connections based on pure IP and tunneling (for a Maximum Transfer Unit—MTU equal to 1500B on the left and 9000B on the right). Connections have been sorted in increasing order, and it can be noticed that the performances are quite acceptable for all connections (always under 5 ms). On both sides of the figures, some nearly comparable values are observed, so to better understand how the RTT evaluations are related in function of time, their pdfs have been considered.

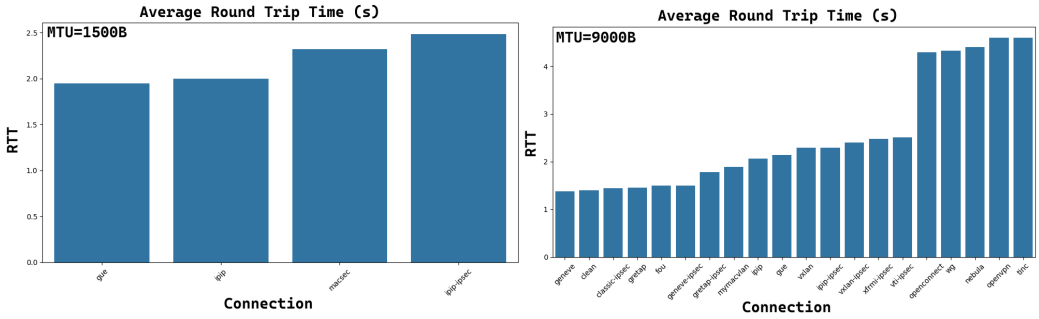

Figure 7. The trend of the average RTT for the considered connections.

Figure 8 shows how the instantaneous RTT is distributed over 10 min of simulations for the first seven tunnels of Figure 7. Their distributions can be perfectly comparable in terms of both means and standard deviations.

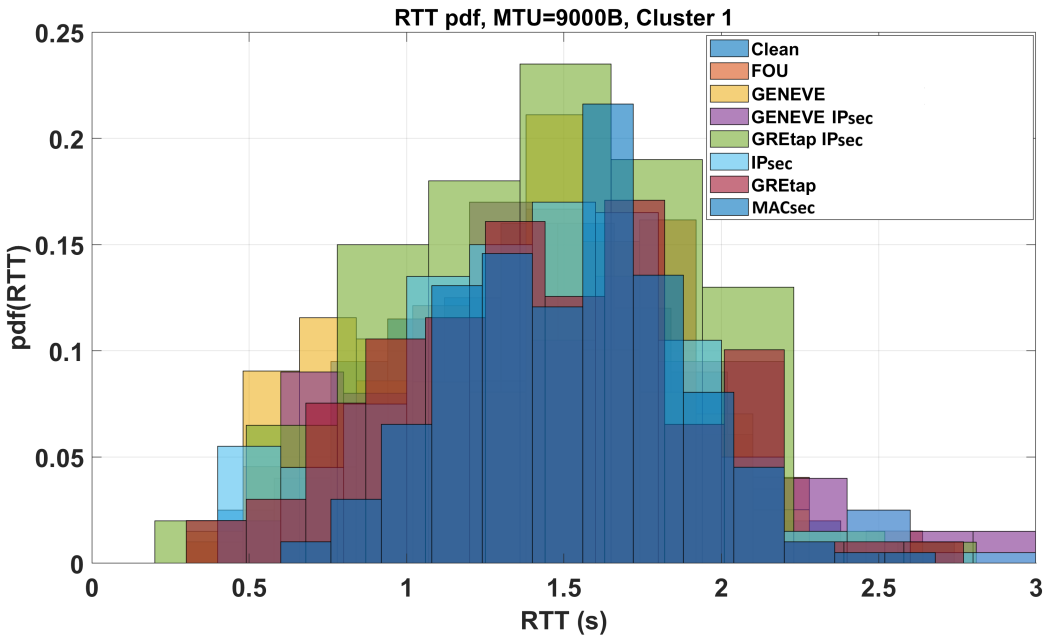

Figure 8. Clustered RTT distribution trend for the first six tunnels (one refers just to a clean IP connection) of Figure 7.

Although there is a slightly higher difference in the mean values, another cluster can be identified, as in Figure 9, for the middle set of tunnels illustrated in Figure 7. Also, the mean and standard deviations are mostly comparable in this case.

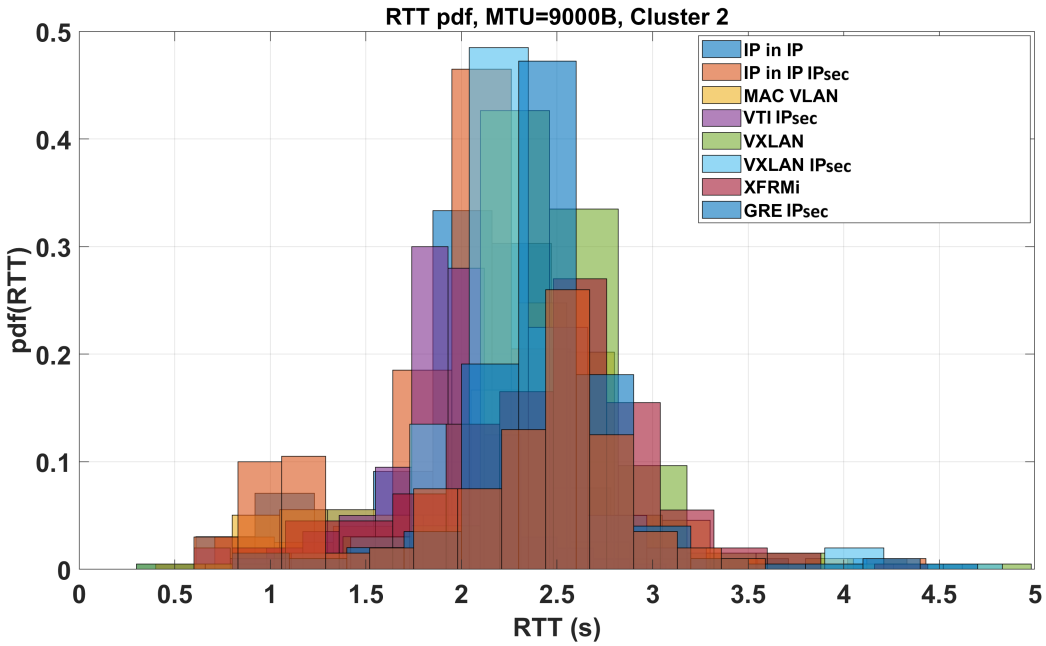

Figure 9. Clustered RTT distribution trend for the central group of tunnels illustrated in Figure 7.

The same considerations for Figures 8 and 9 can be made for Figure 10, where the last tunnels of Figure 7 (right side) have been clustered. We also show how the RTT is distributed over time with MTU = 1500B and how the distributions can also be clustered in this case: Figures 11–13.

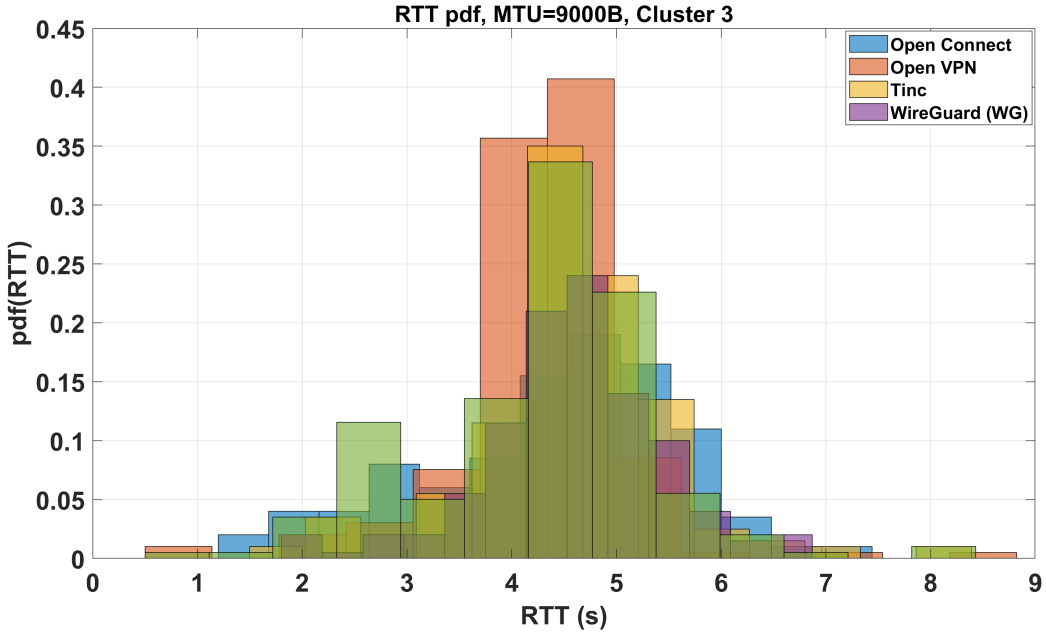

Figure 10. Clustered RTT distribution trend for the central group of tunnels illustrated in Figure 7.

At this point, we can state that for a Clean (no tunneling), FOU, GENEVE, GENEVE IPsec, GREtap, GREtap IPsec, and IPsec (Cluster 1) the average RTT with MTU = 9000B is μ_{RTT} = 1.4231 s, with a standard deviation of σ_{RTT} = 0.4821 s. For GRE IPsec, IP in IP, IP in IP IPsec, MAC VLAN, VTI IPsec, VXLAN, VXLAN IPsec, and XFRMi (Cluster 2) the average RTT with MTU = 9000B is μ_{RTT} = 2.2439 s, with σ_{RTT} = 0.6458 s. For Open Connect, Open VPN, Tinc, and WG (Cluster 3), the average RTT with MTU = 9000B is μ_{RTT} = 4.4304 s, and σ_{RTT} = 0.9753 s.

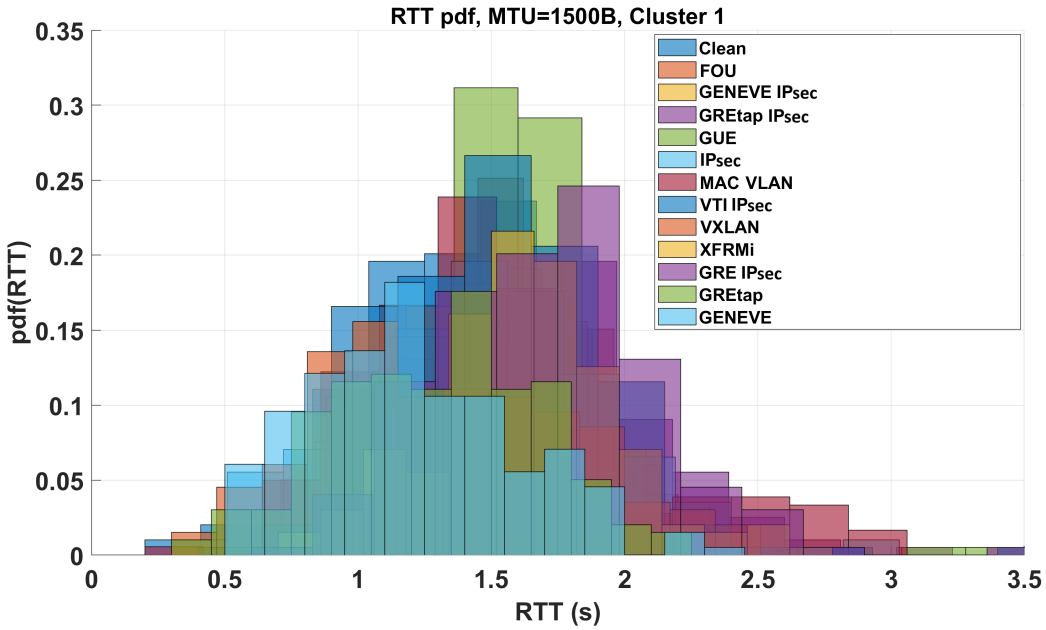

Figure 11. Clustered RTT distribution trend for the central group of tunnels illustrated in Figure 7.

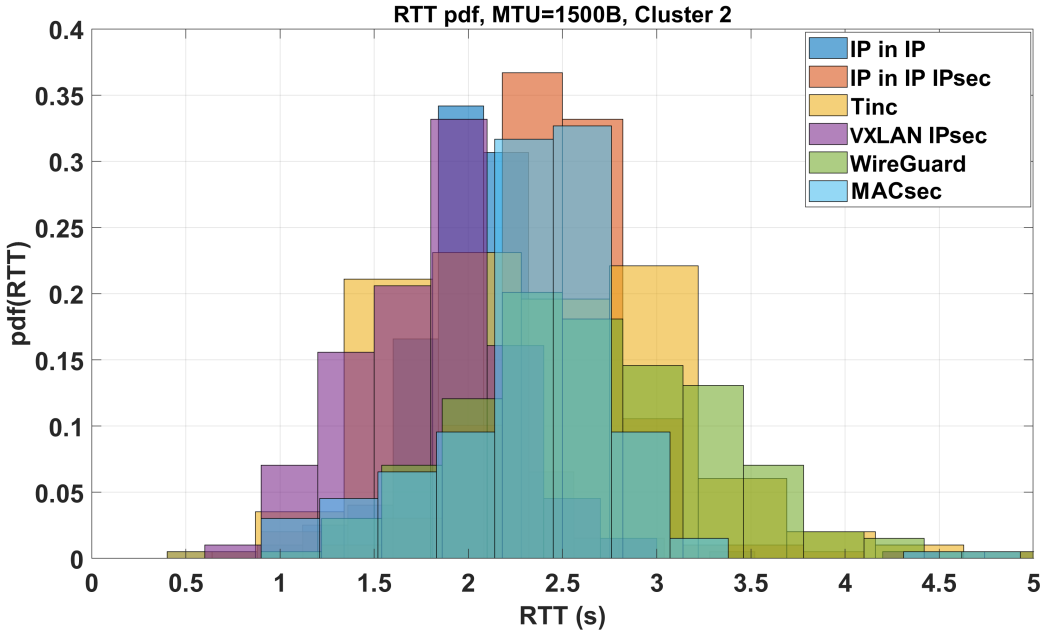

Figure 12. Clustered RTT distribution trend for the central group of tunnels illustrated in Figure 7.

As regards the MTU = 1500B, we found that for Clean, FOU, GENEVE, GENEVE IPsec, GRE IPsec, GREtap, GREtap IPsec, GUE, IPsec, MACVLAN, MACsec, VTI IPsec, VXLAN, and XFRMi (Cluster 1) μ_{RTT} = 1.5247 s and σ_{RTT} = 0.43023 s, for IP in IP, IP in IP IPsec, Tinc, VXLAN IPsec, WireGuard and MACsec (Cluster 2), μ_{RTT} = 2.12041 s and σ_{RTT} = 0.4868 s and for Nebula, OpenConnect and OpenVPN (Cluster 3), μ_{RTT} = 3.22462 s and σ_{RTT} = 0.52786 s. Bold fonts are used to connect different clusters in the MTU function. From the data illustrated above, we can conclude that, independently from the chosen MTU, in terms of RTT, Clean (obvious), FOU, GENEVE, GENEVE IPsec, GREtap, and GREtap IPsec offer the best performance (the needed header length is low and their reaction time is the minimum for MTUs of both 1500B and 9000B). Out of Cluster 1, the RTT worsens in the needed header dimension function.

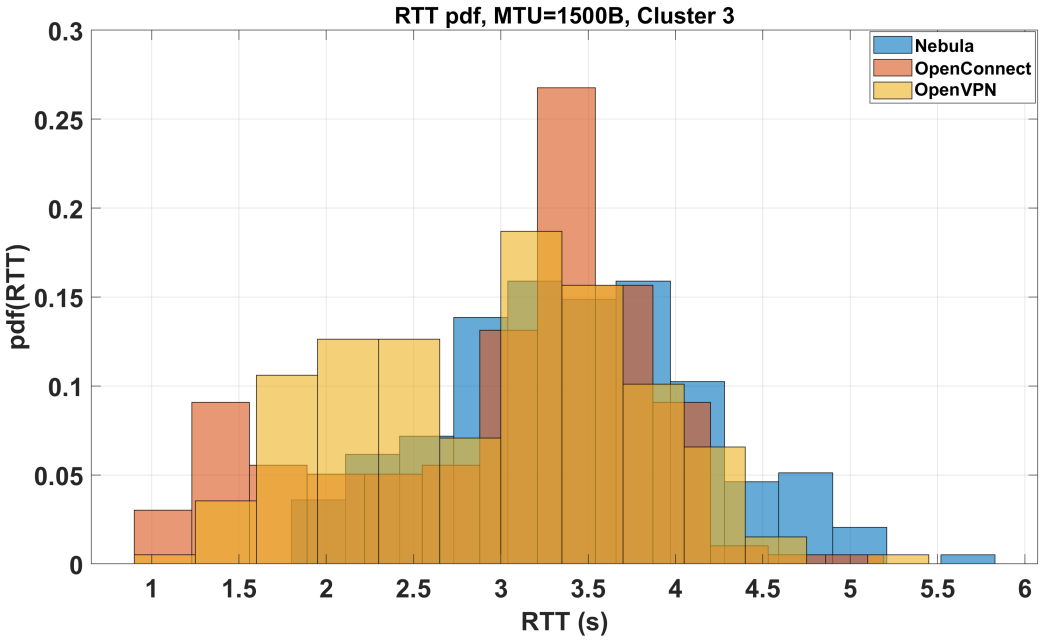

Figure 13. Clustered RTT distribution trend for the central group of tunnels illustrated in Figure 7.

Let us examine how MTU sizes affect network performance in overlay channels and VPNs. Using larger MTU sizes enables the operating system to send fewer but larger packets while maintaining the same network speed. This significantly reduces the processing required by the operating system, provided that the workload allows for the sending of larger messages. However, larger MTU sizes will not have a significant impact if the workload primarily consists of small messages. Using the largest MTU size supported by the adapter and the network is advisable. For example, with ATM adapters, the default MTU size of 9180 bytes is much more efficient than using an MTU size of 1500, which is commonly used by LAN Emulation. In Gigabit and 10 Gigabit Ethernet networks, if all machines have Gigabit Ethernet adapters and no 10/100 adapters are present, it is best to use jumbo frame mode. For instance, connections from server to server within a computer lab can typically be optimized using jumbo frames. The Maximum Transmission Unit (MTU), or link MTU, is measured in bytes and represents the largest packet size that can be sent over a hop without fragmentation. This differs from the Path MTU (PMTU), the smallest MTU among all hops in a path. Iperf assesses IP network bandwidth availability. When packet loss is present, PMTU discovery may occasionally underestimate the MTU. To validate the discovered MTU, conducting multiple tests for consistency is advisable. However, if the discovered MTU matches a recognized standard, it is typically not an inadvertent underestimation. Common values for MTU are shown in Table 4:

Table 4. Common MTU values by link type.

MTU TYPE	MTU SIZE (Bytes)
IP over SONET	4470
Ethernet Jumbo Frames	9000
IP over ATM Ethernet Jumbo Frames	9180
Classic Ethernet	*1500*

The overhead introduced by various tunneling protocols varies depending on the specific protocol and configuration. Focus on Overlay links deployed is shown in Table 5. These overheads should be considered when designing and implementing network tunnels, as they affect the overall payload size and network performance.

Table 5. Size of the headers of the analyzed tunnel protocols and types of headers used.

Tunneling Protocol	Header Type	Header Size (Bytes)	Max Header Size (Bytes)
IPIP	IP	20	20
VTI	IP + VTI	20 + 4	24
GRE	GRE	4 + [0...34]	38
GREtap	Ethernet Frame + GRE	14 + 4	18
FOU	UDP + IP	8 + 20	28
GUE	UDP + GRE	8 + 4 + [0...34]	46
GENEVE	UDP + IP + GENEVE	8 + 20 + 10	38
VXLAN	UDP + IP + VXLAN	8 + 20 + 8 + [0...16]	52
MACVLAN	Ethernet Frame + MACVLAN	14 + 4	18
MACsec	Ethernet Frame + MACsec	14 + [8...30]	44

When utilizing a tunnel interface over IPsec, the overhead includes IPsec headers, the outer IP header for IPsec, and the additional IP header for tunnel encapsulation. The IPsec header comprises the Security Association (SA) information, encryption and authentication headers (if applied), and other IPsec-related metadata. IPIP encapsulation introduces another IP header, encompassing the tunnel endpoints' source and destination IP addresses and any supplementary IP options.

The total overhead for a tunnel over IPsec is the aggregate of the IPsec and tunnel type headers, along with any additional IPsec overhead. This overhead varies based on the specific IPsec configuration (e.g., encryption and authentication algorithms) and tunnel type configuration. A detailed IPsec and tunnel configurations analysis is required to determine the specific overhead. Generally, IPsec overhead ranges from tens to hundreds of bytes, while the IPIP header typically adds twenty bytes.

In Libreswan, the default Maximum Segment Size (MSS) for both tunnel and transport modes is usually derived from the MTU. For tunnel mode, the default MTU is generally 1400 bytes, leading to an MSS of 1360, calculated as MTU minus the IP and TCP header sizes (each 20 bytes for IPv4). For transport mode, where the default MTU is typically 1500 bytes, the MSS is 1460, computed similarly.

These calculations are based on standard IPv4 and TCP header sizes. Additional packet headers, options, or extensions may affect the MSS value. Nonetheless, these calculations provide a general guideline for determining MSS in Libreswan based on default MTU values.

Table 6 summarizes the VPN tunneling protocols used.

Table 6. MTU and MSS sizes for various types of VPNs, along with the transport layers they use.

VPN TYPE	TRANSPORT LAYER	Default MTU SIZE (Bytes)	Default MSS SIZE (Bytes)
OpenVPN	TCP/UDP	1500 bytes	1450 bytes
WireGuard	UDP	1500 bytes	1450 bytes
Tinc	UDP	1500 bytes	1450 bytes
Nebula	UDP	1300 bytes	1260 bytes

Each protocol has its strengths and trade-offs, making them suitable for various network configurations and requirements.

Figures 14 and 15 show the connections performance for TCP and UDP connections, respectively, obtained by the *iperf* command.

The experiments with Iperf and Netperf were conducted as follows. The server (Iperf/Netperf) is started on two endpoints: TCP mode and UDP mode. The MTU is set at the moment of tunnel creation. For IPERF TCP, the client sends 100 Mb of traffic to the server for each negotiated overlay/VPN channel. In IPERF UDP mode, the client sends 100 Mb of traffic to the server for each negotiated overlay/VPN channel with a bandwidth of 100 MB. For NETPERF TCP, a TCP network performance test is performed between the local host and the remote host at 10.0.0.1 for 10 s. During the test, CPU usage is measured on both the local and remote sides. In NETPERF UDP mode, a network performance test

is performed between the local host and the remote host at 10.0.0.1 using UDP for 30 s, measuring the transfer speed in Mbps. The remote side sends UDP packet streams sized at 1024 bytes.

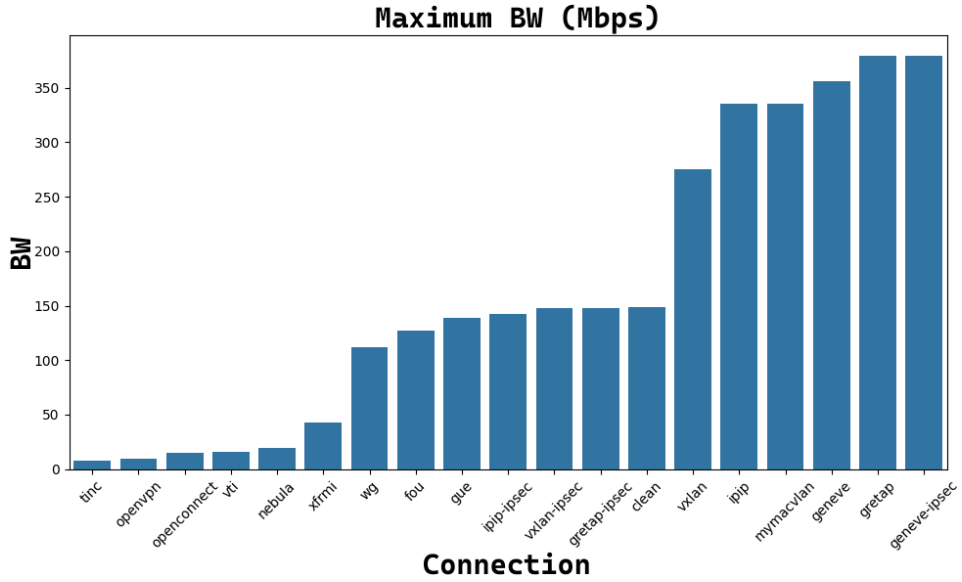

Figure 14. The trend of maximum reachable bandwidth (Mbps) for each TCP connection with the underlying tunnel (MTU = 9000B).

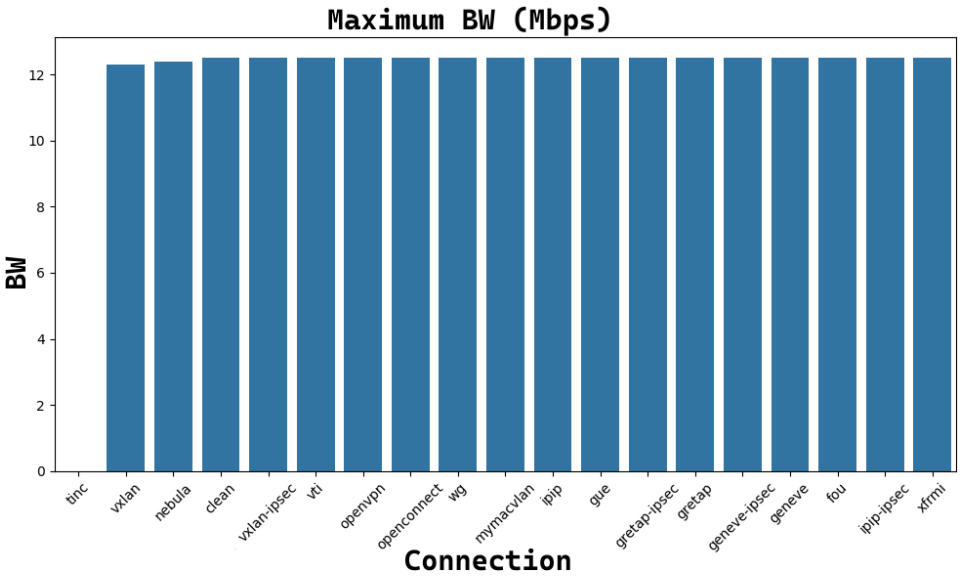

Figure 15. The trend of maximum reachable bandwidth (Mbps) for each UDP connection with the underlying tunnel (MTU = 9000B).

First of all, it can be noticed that *tinc* shows very bad performance and following are some of the reasons why Tinc VPN may experience low performance:

- An MTU that is too high or too low can cause packet fragmentation or inefficiency in packet transfer. A value of 1400 can improve performance.
- If compression is enabled, it can cause extra CPU load.
- Suboptimal TCP/IP parameters can affect performance.
- Using outdated versions of Tinc can lead to performance issues, and Tinc has not been updated since 2021.

In both cases, Tinc connections are completely unsuitable. In the UDP cases, there are no differences in terms of bandwidth, while for layer 4 TCP sockets, OpenVPN, OpenConnect, VTI, Nebula, and XFRMi are unsuitable (very low reachable bandwidth). WG, FOU, GUE, IP in IP IPsec, VXLAN IPsec, and GREtap IPsec performances are comparable to the performances obtained with a Clean connection. The other tunnels outperform the previous ones.

Before going on with the other results, we would like to underline two main aspects:

- Regarding Tinc connections, it is known [30] that Tinc performs better for low MTU sizes (below 1500B). We show the reachable RTT(s) obtained for low MTU sizes for completeness. From Figure 16, it is evident that the RTT is quite acceptable for low values of MTU.
- UDP low reachable bandwidth (as shown in Figure 15, where a bandwidth of 12.4 Mbps is never exceeded): running virtual machines and enabling tunneled connections slow UDP throughput due to extra virtualization overhead and networking configuration issues. Typically, for any UDP packet exceeding 1024 bytes, the Windows network stack delays sending the next packet until it receives a transmit completion interruption.

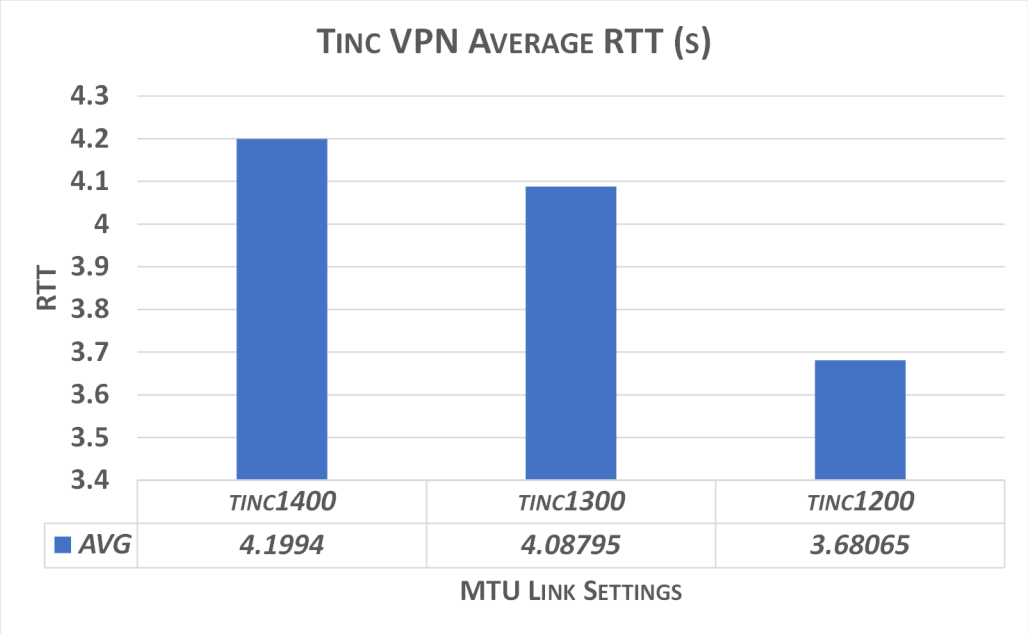

Figure 16. The trend of Tinc average RTT for different MTU lengths.

Optimizing configurations across all layers and ensuring hardware support can help mitigate performance issues.

Figures 17 and 18, shows the reachable throughput for TCP and UDP sessions respectively. It can be noticed that tunneling produces many more effects with the TCP instead of the UDP, it can be seen that the average throughput goes down for Nebula, OpenConnect, OpenVPN, tinc, and XFRMi, while WireGuard throughput is still comparable to the one of a clean connection. Regarding UDP connections, the Netperf application allows us to appreciate better performance for Nebula, OpenConnect, OpenVPN, and Tinc. At the same time, WireGuard and XFRMi can maintain the same performance of a clean connection.

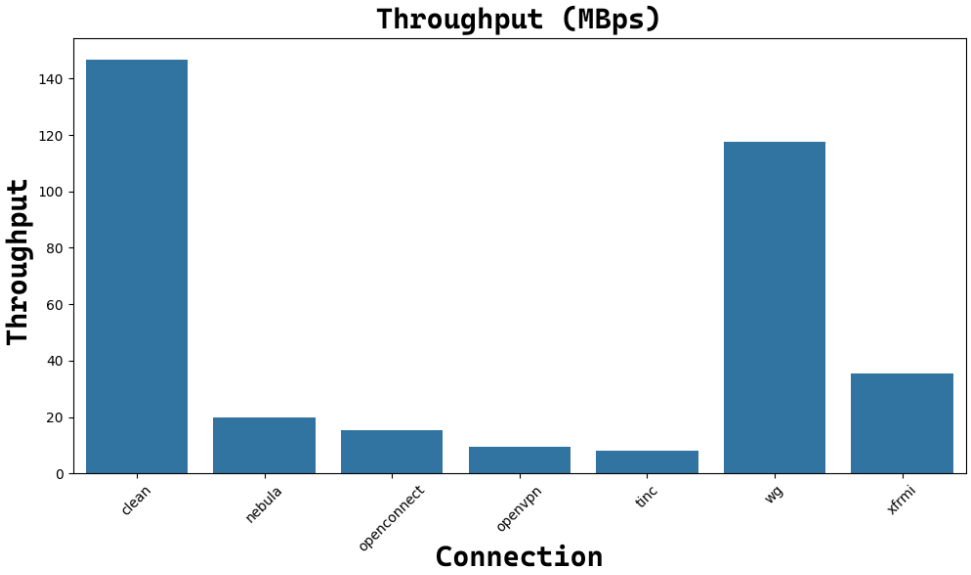

Figure 17. The trend of the average throughput (Mbps) for each TCP connection with only a VPN tunnel (MTU = 9000B).

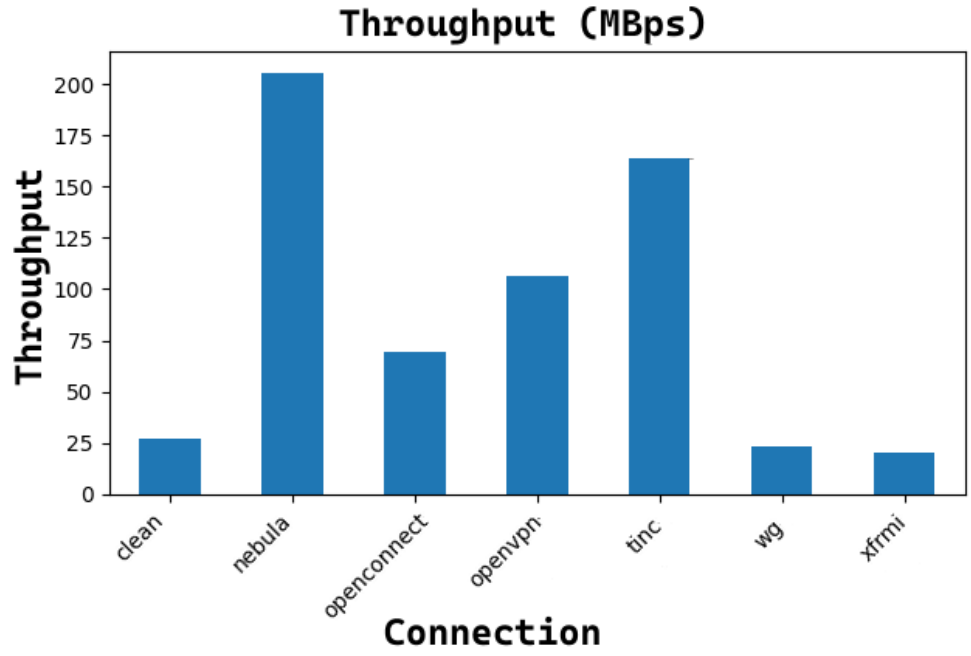

Figure 18. The trend of the average throughput (Mbps) for each UDP connection with only VPN tunnels (MTU = 9000B).

Figures 19 and 20 show the connection performance for TCP and UDP connections, respectively, obtained by the *iperf* command, with an MTU=1500B. Also, in this case, for the TCP case, there are unsuitable tunnels, such as *openconnect*, *vti*, *gre IPSec*, *vxLAN IPSec*, *xfrmi*, *IP IP IPSec*, *Gretap IPSec*, and the *clean* connection. Then, another cluster of tunnels provides an average performance, like *vxlan* and *Geneve*. The best results are obtained with *Geneve IPSec*, *gre*, *gretap*, *FOU*, *gue*, *IP in IP*, and *myMACVLAN*. As for the MTU = 9000B case, UDP saturates all connections for the same reasons explained in the previous case with MTU = 9000B.

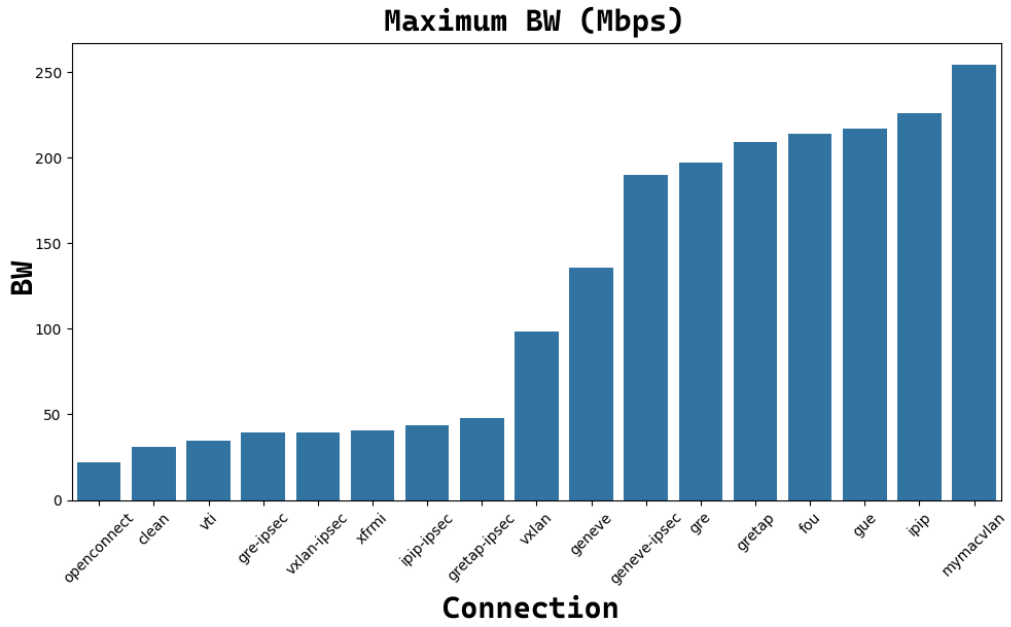

Figure 19. The trend of maximum reachable bandwidth (Mbps) for each TCP connection with the underlying tunnel (MTU = 1500B).

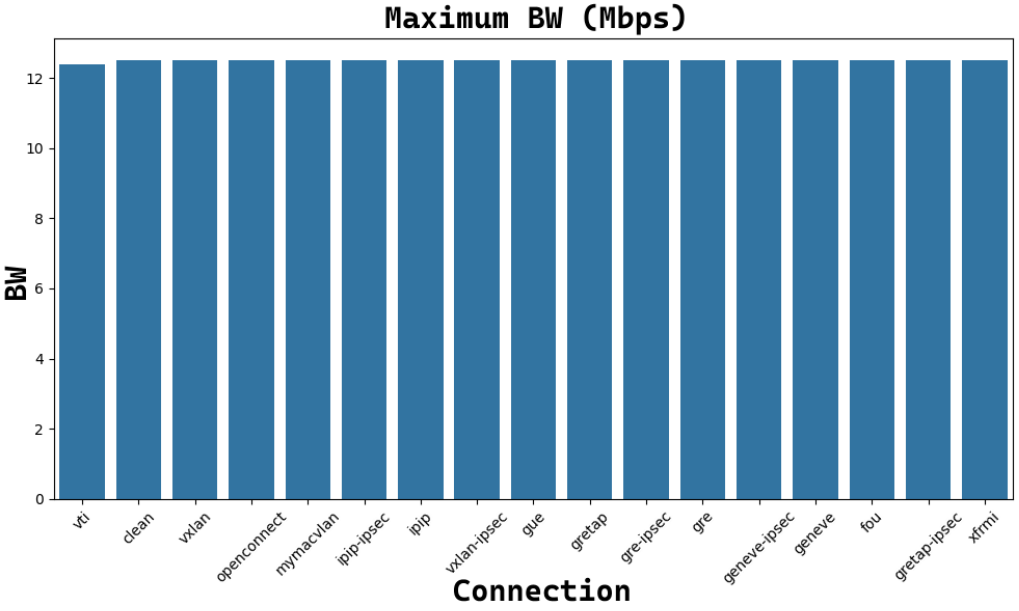

Figure 20. The trend of maximum reachable bandwidth (Mbps) for each UDP connection with the underlying tunnel (MTU = 1500B).

Figures 21 and 22 show the obtained performance with the *netperf* command for TCP and UDP tunnels, respectively, with MTU=1500B. The obtained values are comparable in the TCP case, while for the UDP case, only the *OpenConnect* tunnel exhibits a very good throughput.

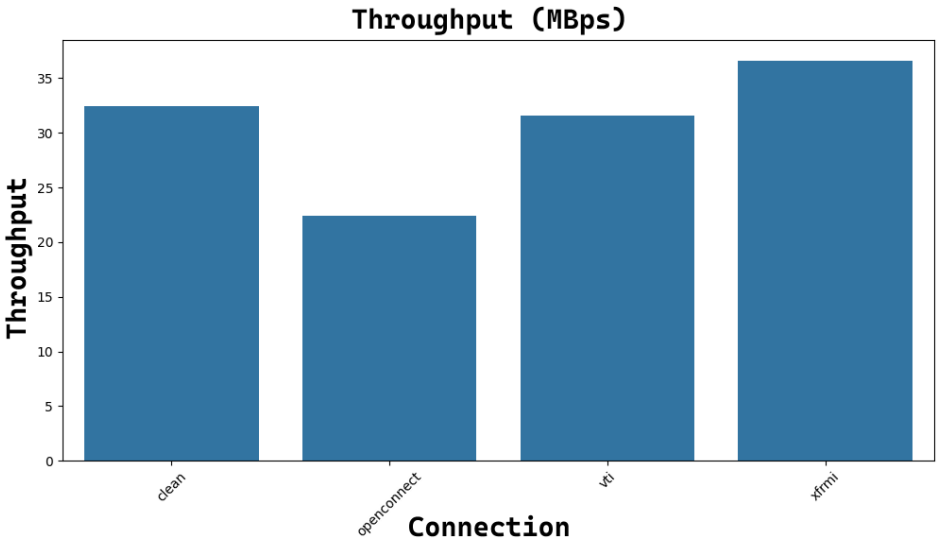

Figure 21. The average reachable throughput (MBps) for each TCP connection with the underlying tunnel (MTU = 1500B).

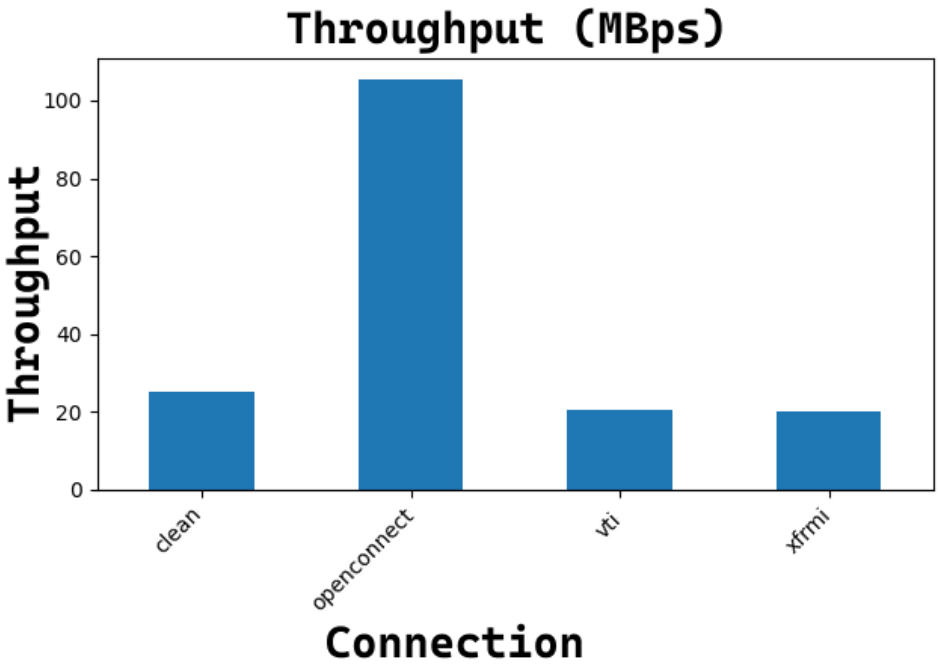

Figure 22. The average reachable throughput (MBps) for each UDP connection with the underlying tunnel (MTU = 1500B).

6. Conclusions and Future Works

In this work, we provided several details about VPN and Overlay connections in TCP/IP networks. In particular, we focused on analyzing several tunnels from different points of view, implementing them into virtualized environments, and giving different information on how they can be implemented. This work's second part has been dedicated to analyzing the obtainable results regarding RTT, maximum bandwidth, and throughput. One of the main results is the possibility of clustering the tunnels in the function of the reachable RTT, thus providing a wide variety of choices that respect a given RTT distribution, depending on the desired results. The second analysis that has been illustrated regards the performance of tunneling for TCP and UDP connection: we have shown how, for UDP, all the connections saturate, while for TCP, there are some preferred tunnels. The same trend has been found for the throughput analysis.

One of the future works will certainly be to translate this study onto the Proxmox platform to evaluate the performance achievable through systems managed through this hypervisor. Proxmox VE optimizes the network and CPU management layers to deliver high performance and efficiency in virtualized environments. Here is an emphasis on these optimizations: Proxmox VE supports a wide array of advanced networking technologies crucial for creating flexible and high-performance virtual networks. These include:

- FOU (Foo Over UDP): efficient encapsulation of IP in UDP, reducing overhead.
- GUE (Generic UDP Encapsulation): offers flexibility for different protocols with minimal latency.
- GRE (Generic Routing Encapsulation) and GREtap: ensures robust tunneling of various network protocols and Ethernet frames, optimizing traffic flow.
- IPIP (IP-in-IP): simplifies tunneling with minimal processing overhead.
- GENEVE and VXLAN: provide scalable solutions for network virtualization, enhancing performance in overlay networks.
- MACsec: secures data link layer communications, maintaining high throughput with encryption.
- OpenVPN and WireGuard: deliver secure and efficient VPN solutions, with WireGuard specifically known for its minimal CPU usage and high performance.
- IPsec (VTI, XFRMi, Classic): ensures secure communications with optimized encryption and tunneling techniques.
- Tinc and Nebula: offer mesh networking solutions that scale efficiently while maintaining security.

Proxmox VE enhances CPU management through various techniques to ensure that virtual machines (VMs) and containers run efficiently:

- KVM (Kernel-based Virtual Machine): utilizes hardware virtualization extensions (Intel VT-x and AMD-V) to ensure near-native performance for VMs.
- CPU Pinning: allows binding of VMs to specific CPU cores, reducing context switching and optimizing performance.
- NUMA (Non-Uniform Memory Access) Awareness: Proxmox can optimize memory and CPU allocation across NUMA nodes, improving access times and overall VM performance.
- Resource Limits and Quotas: enable setting limits and quotas for CPU usage, ensuring fair distribution of resources and preventing any single VM from monopolizing CPU resources.
- Dynamic Resource Allocation: Proxmox dynamically adjusts resources based on current workload demands, optimizing CPU and memory usage.
- CPU Hotplug: allows adding CPUs to running VMs without downtime, providing flexibility and scalability.
- Integration with cgroups and namespaces: ensures fine-grained control over resource allocation and container isolation, optimizing CPU usage.

Proxmox VE's optimizations in network and CPU management layers ensure that virtualized environments operate with high efficiency and performance. The comprehensive support for advanced networking technologies combined with robust CPU management techniques makes Proxmox VE a powerful platform for modern data centers and cloud environments.

Author Contributions: Conceptualization, A.F.G., P.F. and D.M.; methodology, E.G.; software, A.F.G.; validation, A.F.G., D.M. and E.G.; data curation, A.F.G. and P.F.; writing—original draft preparation, A.F.G. and P.F.; writing—review and editing, A.F.G. and P.F.; supervision, A.F.G. All authors have read and agreed to the published version of the manuscript.

Funding: The research by P.F. leading to a part of the published results was supported by the European Union within the REFRESH project – Research Excellence For Region Sustainability and High-tech Industries ID No. CZ.10.03.01/00/22 003/0000048 of the European Just Transition Fund.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
AH	Authentication Header
EAP	Extensible Authentication Protocol
ESP	Encapsulating Security Payload
GDPR	General Data Protection Regulation
HW	Hardware
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IoT	Internet of Things
IPsec	IP Security
ISAKMP	Internet Security Association and Key Management Protocol
MQTT	Message Queue Telemetry Transport
NAT	Network Address Translation
OTP	On-Time Password
PSK	Pre Shared Key
PSTN	Public Switched Telephone Network
RTT	Round Trip Time
RW	Road Warriors
SAD	Security Association Database
SPD	Security Policy Database
SSL	Secure Sockets Layer
SW	Software
TLS	Transport Layer Security
UTP	Unshielded Twisted Pair
VPN	Virtual Private Network

References

1. Khanvilkar, S.; Khokhar, A. Virtual private networks: An overview with performance evaluation. *IEEE Commun. Mag.* **2004**, *42*, 146–154. [CrossRef]
2. Alshalan, A.; Pisharody, S.; Huang, D. A survey of mobile VPN technologies. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 1177–1196. [CrossRef]
3. Gentile, A.F.; Fazio, P.; Miceli, G. A Survey on the Implementation and Management of Secure Virtual Private Networks (VPNs) and Virtual LANs (VLANs) in Static and Mobile Scenarios. *Telecom* **2021**, *2*, 430–445. [CrossRef]
4. Troia, S.; Mazzara, M.; Moreira Zorello, L.M.; Maier, G. Performance Evaluation of Overlay Networking for delay-sensitive services in SD-WAN. In Proceedings of the 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Athens, Greece, 7–10 September 2021; pp. 150–155. [CrossRef]
5. Farinacci, D. Generic Routing Encapsulation. RFC 2784, 2000. Available online: https://datatracker.ietf.org/doc/rfc2784/ (accessed on 23 June 2024).
6. Lammle, T. *Virtual Private Networks (VPNs)*; 2020; pp. 433–450. Available online: https://www.researchgate.net/publication/338788069_Virtual_Private_Networks_VPNs (accessed on 2 June 2024).
7. Zhang, L.; Wang, Y.; Liang, S.; Jin, R. Container network architecture and performance analysis of Macvlan and IPvlan. In Proceedings of the 2022 International Conference on Education Innovation and Modern Management (EIMM 2022), Chengdu, China, 16–18 September 2022; Volume 166. [CrossRef]
8. Mao, H.; Zhu, L.; Qin, H. A Comparative Research on SSL VPN and IPSec VPN. In Proceedings of the 2012 8th International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, China, 21–23 September 2012; pp. 1–4. [CrossRef]
9. Thomson, M.; Turner, S. *Using TLS to Secure QUIC*; Internet-Draft draft-ietf-quic-tls-31, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2019.
10. Wood, C.A.; Enghardt, R.; Pauly, T.; Perkins, C.; Rose, K. *A Survey of Transport Security Protocols*; Internet-Draft draft-ietf-taps-transport-security-05, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2019.

11. Pereira, R.; Beaulieu, S. *Extended Authentication within ISAKMP/Oakley (XAUTH)*; Internet-Draft draft-ietf-ipsec-isakmp-xauth-06, Work in Progress; Internet Engineering Task Force : Fremont, CA, USA, 1999.
12. Smyslov, V.; Weis, B. *Group Key Management Using IKEv2*; Internet-Draft draft-ietf-ipsecme-g-ikev2-06, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2022.
13. Cicirelli, F.; Gentile, A.F.; Greco, E.; Guerrieri, A.; Spezzano, G.; Vinci, A. An Energy Management System at the Edge based on Reinforcement Learning. In Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Prague, Czech Republic, 14–16 September 2020; pp. 1–8. [CrossRef]
14. Ezra, P.; Misra, S.; Agrawal, A.; Jonathan, O.; Maskeliunas, R.; Damaševičius, R. Secured Communication Using Virtual Private Network (VPN). In *Cyber Security and Digital Forensics*; UWE Bristol: Bristol, UK, 2022; pp. 309–319. [CrossRef]
15. Mahmmod, K.F.; Azeez, M.M.; Ahmed, M.A. IPsec Cryptography for Data Packets Security within VPN Tunneling Networks Communications. In Proceedings of the 2020 International Conference on Electrical Engineering and Informatics (ICELTICs), Aceh, Indonesia, 27–28 October 2020; pp. 1–8. [CrossRef]
16. Wouters, P. *Deprecation of IKEv1 and Obsoleted Algorithms*; Internet-Draft draft-ietf-ipsecme-ikev1-algo-to-historic-06, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2022.
17. Aung, S.T.; Thein, T. Comparative Analysis of Site-to-Site Layer 2 Virtual Private Networks. In Proceedings of the 2020 IEEE Conference on Computer Applications(ICCA), Yangon, Myanmar, 27–28 February 2020; pp. 1–5. [CrossRef]
18. Gont, F. *Layer 3 Virtual Private Network (VPN) Tunnel Traffic Leakages in Dual-Stack Hosts/Networks*; RFC 7359; Internet Engineering Task Force: Fremont, CA, USA, 2014. [CrossRef]
19. Sanchez, D.; García, M.A. *A Simple SCCP Tunneling Protocol (SSTP)*; Internet-Draft draft-sanchez-garcia-SSTP-v1r0-00, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 1999.
20. Patel, D.B.V.; Aboba, D.B.D.; Dixon, W.; Zorn, G. *Securing L2TP Using IPSEC*; Internet-Draft draft-ietf-pppext-l2tp-security-05, Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 1999.
21. Haga, S.; Esmaeily, A.; Kralevska, K.; Gligoroski, D. 5G Network Slice Isolation with WireGuard and Open Source MANO: A VPNaaS Proof-of-Concept. *arXiv* **2020**. [CrossRef]
22. Ajiya, A.; Idriss, U. Performance Evaluation of IPSEC-VPN on Debian Linux Environment General Terms. 2019. Available online: https://www.researchgate.net/publication/331802877_Performance_Evaluation_of_IPSEC-VPN_on_Debian_Linux_Environment_General_Terms (accessed on 23 June 2024).
23. Gentile, A.F.; Macrì, D.; Rango, F.D.; Tropea, M.; Greco, E. A VPN Performances Analysis of Constrained Hardware Open Source Infrastructure Deploy in IoT Environment. *Future Internet* **2022**, *14*, 264. [CrossRef]
24. Sun, S.H. The advantages and the implementation of SSL VPN. In Proceedings of the 2011 IEEE 2nd International Conference on Software Engineering and Service Science, Beijing, China, 15–17 July 2011; pp. 548–551. [CrossRef]
25. Fei, C.; Kehe, W.; Wei, C.; Qianyuan, Z. The Research and Implementation of the VPN Gateway Based on SSL. In Proceedings of the 2013 International Conference on Computational and Information Sciences, Shiyang, China, 21–23 June 2013; pp. 1376–1379. [CrossRef]
26. Libreswan. Available online: https://libreswan.org/ (accessed on 20 June 2022).
27. Strongswan. Available online: https://www.strongswan.org/ (accessed on 20 June 2022).
28. Openwrt. Available online: https://openwrt.org/ (accessed on 20 June 2022).
29. Mazoni;Olivo, B.; Pan, A. Internet of Things: State-of-the-art, Computing Paradigms and Reference Architectures. *IEEE Lat. Am. Trans.* **2022**, *20*, 49–63. [CrossRef]
30. Guus, S. Tinc VPN, 2024. Available online: http://www.tinc-vpn.org/git/browse?p=tinc;a=log;h=refs/heads/1.1 (accessed on 23 June 2024).

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.