Received: 12 July 2023 | Revised: 10 December 2023 | Accepted: 25 February 2024

DOI: 10.1002/spe.3329

RESEARCH ARTICLE

WILEY

Full-mesh VPN performance evaluation for a secure edge-cloud continuum

Vojdan Kjorveziroski[1] | **Cristina Bernad**[2] | **Katja Gilly**[2] | **Sonja Filiposka**[1]

[1]Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, North Macedonia

[2]Department of Computer Engineering, Miguel Hernández University (Elche), Alicante, Spain

Correspondence
Cristina Bernad, Department of Computer Engineering, Miguel Hernández University (Elche), Avenida de la Universidad, Alicante 03202, Spain.
Email: cbernad@umh.es

Funding information
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University

Abstract

The recent introduction of full-mesh virtual private network (VPN) solutions which offer near native performance, coupled with modern encryption algorithms and easy scalability as a result of a central control plane have a strong potential to enable the implementation of a seamless edge-cloud continuum. To test the performance of existing solutions in this domain, we present a framework consisted of both essential and optional features that full-mesh VPN solutions need to support before they can be used for interconnecting geographically dispersed compute nodes. We then apply this framework on existing offerings and select three VPN solutions for further tests: Headscale, Netbird, and ZeroTier. We evaluate their features in the context of establishing an underlay network on top of which a Kubernetes overlay network can be created. We test pod-to-pod TCP and UDP throughput as well as Kubernetes application programming interface (API) response times, in multiple scenarios, accounting for adverse network conditions such as packet loss or packet delay. Based on the obtained measurement results and through analysis of the underlying strengths and weaknesses of the individual implementations, we draw conclusions on the preferred VPN solution depending on the use-case at hand, striking a balance between usability and performance.

KEYWORDS

edge-cloud continuum, Kubernetes, orchestration, virtual private networks, Wireguard, ZeroTier

1 | INTRODUCTION

As the Internet adoption levels have grown overtime, so have the requirements for secure communication between remote nodes over public and untrusted networks. One popular way of satisfying this requirement is through the use of various virtual private network (VPN) solutions, capable of establishing trusted tunnels over untrusted networks.[1] While traffic

Abbreviations: ACLs, access control lists; API, application programming interface; CNI, container network interface; DERP, designated encrypted relay for packet; IdP, identity providers; KNB, Kubernetes network benchmark; LAN, local area network; MTU, maximum transmission unit; NAT, network address translation; PKI, private key infrastructure; QoS, Quality of Service; SaaS, Software as a Service; VPN, virtual private network.

encapsulation solutions providing tunneling capabilities have existed for decades, they do not inherently offer encryption, a core feature of modern VPNs today, thus requiring upper layer protocols to design and incorporate strategies preventing eavesdropping.

When choosing a new VPN solution, a number of things need to be taken into account, ranging from its security features (supported encryption algorithms), ease of use, performance, the underlying transport protocol which has been chosen (TCP or UDP), supported computing platforms, and supported connection topologies. The majority of VPN products today are cross platform, offering support for both desktop and mobile operating systems. When it comes to transport protocols, usually UDP is the preferred choice when optimizing for performance, due to the drawbacks of tunneling regular TCP traffic from an application layer protocol such as HTTP over a TCP tunnel.[2] An exception to this are networks with restrictive policies, where traffic is severely filtered due to security reasons, and only a handful of common TCP ports are open for outbound communication. An additional aspect which also plays a role in the overall performance of the VPN is its software architecture, and whether it supports multi-threading or it is limited to a single thread.[3,4] Some VPN solutions have kernel modules available for the most popular operating systems,[5] while others trade the performance improvements of running in kernel space for a richer feature set running in user-space.[6]

The most popular VPN protocols today are OpenVPN[7] and Wireguard,[8] both being open-source software released under a permissive license. As a result of this, many of the currently available commercial VPN solutions in fact incorporate either one of them or both. Both OpenVPN and Wireguard have native applications and are free to implement and use on private infrastructure. As their manual configuration does not scale beyond dozens of devices, automated tooling or additional supporting software is needed. In the case of OpenVPN, connection establishment can either be done using shared keys or certificates, with the latter requiring a full Private Key Infrastructure (PKI) in place.[9] Wireguard's initial configuration is somewhat simpler, with a process involving the generation of key-pairs and exchanging public keys between endpoints, a procedure reminiscent of SSH public key authentication.[10]

Leveraging Wireguard's simple key exchange along with its improved performance due to its multithreaded architecture, recently a number of as-a-service products have appeared, aiming to solve the scaling issues associated with interconnecting many devices.[11–14] These solutions usually transform the original decentralized key exchange with a central coordination server which acts as a middleman in exchanging the public keys between peers in the same network. This comes with a guarantee that no eavesdropping is possible, as the private key needed for decryption never leaves the device where it was generated. In this setup any node can be connected with an arbitrary number of other nodes and the addition of a new node does not require any manual effort in terms of key exchange or configuration. Thus, it becomes feasible to replace the hub-and-spoke connection topology commonly associated with traditional VPN solutions, with a better performing and redundant full-mesh topology. With all nodes being directly connected to each other and traffic not traversing intermediary transit nodes capable of decrypting it, the full-mesh connectivity eliminates communication bottlenecks and improves security.

The above-mentioned improvements can be utilized in various scenarios, ranging from establishing site-to-site connectivity between remote locations to providing secure client access to corporate infrastructure for individual users.[15] It also has incredible potential for use in edge computing and the establishment of an edge-cloud continuum, where remote sites can be seamlessly interconnected both between each other and with the upstream cloud data centers. The full-mesh topology guarantees minimal latency, being performance dependent only on the hardware characteristics of the device and the efficiency of the VPN protocol. Modern network address translation (NAT) traversal techniques, including relaying, can also ensure VPN connectivity between otherwise unreachable nodes located behind restrictive firewalls.

Inspired by the potential implications of recent edge-cloud developments and the introduction of multiple novel open-source VPN products which can be completely self-hosted without relying on third-party commercial infrastructure, the goal of this paper is to evaluate the applicability and performance of full-mesh VPN solutions when used as an underlay network for the Kubernetes container orchestrator. The underlay network is to be used by container network interface (CNI) plugins which would set up an overlay network on top,[16] transparently interconnecting multiple Kubernetes nodes regardless to physical network or geographical location. To this end, the main contributions of this paper are:

- Review popular full-mesh VPN solutions, providing a framework based on supported features;
- Define requirements which need to be satisfied by full-mesh VPN solutions so that they can be used as an underlay network for orchestration systems, including Kubernetes;
- Benchmark three full-mesh VPN solutions with built-in NAT traversal capabilities (Headscale, Netbird, ZeroTier) when used as an underlay network for the Calico CNI in a Kubernetes environment by comparing to a defined baseline;

- Evaluate real-world connectivity issues and the robustness of the VPN solutions by simulating adverse network conditions, inducing network delay and packet loss, reporting on the measured performance hits;
- Compare the Kubernetes application programming interface (API) response times with and without using a VPN as the underlay network;
- Thorough analysis of the obtained results with guidelines on choosing the best performing VPN solution depending on the use-case.

The rest of this article is organized as follows: in Section 2, we discuss related work to the topic at hand and report on the results of other studies which have concerned themselves with benchmarking VPN performance in various environments. We then proceed with Section 3 where we define the benchmarking methodology, the criteria for selecting the VPN solutions, briefly explaining their specifics, and present the overall execution strategy for the various tests. We discuss the obtained results in Section 4 and summarize our findings in Section 5, elaborating on the advantages as well as drawbacks of each of the tested solutions, and their applicability in different scenarios. We conclude the article with Section 6, drawing conclusions and outlining ideas for future research.

2 | RELATED WORK

The study of tunneling protocols and secure connectivity options, including VPNs has been a popular research area throughout the years. A number of papers have discussed performance aspects of various VPN technologies available at the time of their writing. One such example is Reference 17 which compares the performance of OpenVPN versus IPSec, concluding that IPSec achieves more favorable results in the majority of the tests. While OpenVPN is commonly included in VPN performance studies, newer articles also focus on a more recent introduction to this space, Wireguard. Dekker et al. in Reference 18 describe the performance differences between three VPN protocols: StrongSwan, OpenVPN and Wireguard in a 1Gbit/s network environment. Their findings show that StrongSwan and OpenVPN have the best results when it comes to tunneling UDP traffic, while Wireguard's kernel implementation takes the lead when it comes to connection initialization speed. The user space implementation of Wireguard (Wireguard Go) shows the highest CPU utilization among all tested solutions. The results presented in Reference 19 by Goethals et al. further validate the performance benefits of Wireguard, where it is the most performant solution among those tested (OpenVPN, Wireguard, ZeroTier, Tinc, SoftEther). The authors also include benchmarks focusing on the scalability aspects of the various solutions, testing them with varying number of clients, as well as their robustness by measuring their failure rates.

Recognizing the potential performance advantages of Wireguard on one hand, but also the manual and often time consuming configuration of peers in large topologies on the other, Paillisse et al. describe a centralized control plane for Wireguard, which can facilitate automatic key exchange between participating nodes in a full-mesh network.[20] This eases the configuration process, while not compromising security, since the presented Wireguard control plane only has access to the public keys of all the peers (and not the private ones), along with basic connectivity information, such as IP address and port.

The possibility of configuring full-mesh topologies using Wireguard and thus avoiding the performance penalties associated with alternative hub-and-spoke VPN solutions is well suited for the deployment of geographically distributed computing infrastructures. The authors of Reference 21 evaluate Wireguard's performance when used as a connectivity layer for a distributed Kubernetes cluster, measuring the response time of a web application hosted within it. While the described scenario is successfully validated, no discussion is available on whether alternative VPN solutions have been considered, or whether evaluation of other performance aspects despite application response times was performed (such as throughput or Kubernetes API response times). Wang et al. also validate the idea of distributed Kubernetes clusters on top of Wireguard. However, their main focus is not the VPN connectivity itself,[22] but rather the design of the overall architecture of the proposed solution. The author of Reference 23 also extends a Kubernetes cluster to the edge by incorporating low powered devices through the use of Netmaker, a VPN solution based on Wireguard which leverages the seamless on-boarding experience of new devices provided by a centralized control plane. Wireguard is also utilized as the chosen connectivity option for allowing federation of various Kubernetes clusters described by Falcone et al.[24] through the use of virtual kubelets[25] acting as an interface for interacting with the remote federated clusters. In some cases, new VPN protocols are proposed as well, as is the example with the recent paper describing EdgeVPN.[26] Notable is the NAT traversal support using the standard combination of STUN and TURN servers, but there is no

mention whether connection relaying is only supported via UDP or TCP as well, an important factor when it comes to restrictive networks where UDP traffic is partially or completely blocked. The authors have validated the applicability of EdgeVPN by creating a distributed Kubernetes cluster using the Flannel CNI as an overlay network over the EdgeVPN underlay.

Robust and performant full-mesh VPN solutions which offer seamless node lifecycle management (addition, update, and removal of nodes) have a tremendous potential to impact the future development of a true cloud-edge continuum, allowing data to be initially preprocessed at the edge, before sending it through a secure communication channel for long term storage and more complex computation in the cloud.[27] Support for various NAT traversal techniques[28] along with optional traffic relaying over TCP would also enable edge nodes to be placed in restrictive networks, allowing communication with the cloud as well as other nearby edge nodes even in cases where direct connectivity would otherwise not have been possible. Secure connectivity with all of the previously mentioned improvements can also have an impact on facilitating stateful workloads to be dynamically migrated between the cloud and the edge, utilizing novel container migration techniques[29] together with a VPN backed underlay network.

In conclusion, a number of papers that compare VPN performance between various protocols already exist and initial validations of the idea of using a VPN underlay network for establishing distributed compute infrastructures have been presented. However, to the best of our knowledge, there is no single work which compares the more recent options for establishing VPN full-mesh topologies in this space and evaluates their performance together with NAT traversal and traffic relaying options from a Kubernetes perspective. With this paper, we aim to bridge this gap, evaluating not only the throughput performance of recently introduced contenders, but also their robustness when faced with packet loss and traffic delay. All of these aspects are evaluated both when a direct connection is available between the participating nodes, as well as when traffic relaying alternatives need to be utilized.

3 | METHODOLOGY

The development of new VPN protocols and solutions, together with the performance evaluation of existing ones has always been a vibrant research area, driven by the continuous increase in compute performance, design of new encryption algorithms, and improvements to network capacity. Recently, interest in this research field has increased even further as a result of the introduction of the Wireguard VPN protocol, allowing high performance without complex configuration.

Taking into account the large number of VPN solutions which are currently available, a well-defined and structured evaluation methodology is required to achieve the end-goal of determining the most well suited ones for interconnecting distributed Kubernetes clusters. In this section we elaborate on the approach taken in our study of such VPN options, starting with the selection criteria for the VPN software in Section 3.1. We outline the essential features that VPN solutions should provide in such a context, as well as additional accompanying functionality which although not essential, would be a welcome addition. We then proceed with the description of the testing environment in Section 3.2, where we provide more information on the architecture of the evaluated VPN solutions, along with details on how the baseline has been established. We conclude this section with Section 3.3, discussing the employed testing strategy both for the throughput tests, as well as for measuring the Kubernetes API response times.

All of the output from this research, starting from the source code for the benchmarking tools, to the results, and finally their analysis is open-sourced and published under a permissive license,* available for reuse by other researchers.

3.1 | VPN software selection

The selection criteria for the VPN software to be evaluated is divided into two parts: essential features which are required for the VPN solution to be used in the context of dispersed Kubernetes clusters; complementary features that improve usability in versatile scenarios.

The essential features which we have defined, and which need to be supported by a given VPN solution before it is included in the tests are:

*The data is available on https://github.com/korvoj/vpn-performance.

- Must be completely open-sourced, without any proprietary modules, and provide the possibility for deployment on private infrastructure. This aspect is required to ensure full control of both the traffic between the nodes, as well as the control plane, avoiding any potential vendor lock-in in the process.
- Must be actively maintained, with a new release in the last 6 months, ensuring on-time security updates and bug fixes.
- Must support full-mesh topology, where each node would connect to all other nodes where possible, thus avoiding bottlenecks and maintaining low communication latency.
- Must provide an option for unattended registration of new nodes and their addition to the mesh, thus enabling dynamic scaling of the underlying infrastructure and extension with extra compute nodes when required.
- Must support access control lists (ACLs) for controlling what nodes can communicate between themselves. This feature allows the same VPN solution together with a single control plane to be used for interconnecting multiple, independent, globally distributed infrastructures. As a result, multiple isolated meshes between devices can be configured, with devices being restricted with whom they can communicate and establish VPN connections based on the configured ACLs.
- Must support multiple architectures, with clients for both x86 and ARM devices. This would allow the VPN solution to be used both for interconnecting more powerful x86 servers, as well as power efficient ARM devices located at the edge of the network.
- Must provide advanced connectivity options for interconnecting devices located behind restrictive firewalls. This includes NAT traversal support, as well as full relaying (over both TCP and UDP) through an intermediary node which is placed at a strategic location with low latency to the other nodes and accepting remote connections. The relaying should be implemented in such a way so that the intermediary node does not see the transit traffic in clear text, preserving the end-to-end encryption between the nodes.

The set of features which although not essential would offer additional possibilities are:

- Granular ACL support. It is desirable to not only define whether bidirectional communication between the nodes is allowed or not, but also on what ports, as well as an option to explicitly specify the traffic direction.
- Support for subnet routers, where a node can be used as a gateway for accessing isolated notes from the local area network (LAN) behind that particular node. This would allow access to devices which are not able to become part of the mesh due to their constrained performance or exotic operating systems, with the drawback of the gateway node being able to decrypt the transit traffic.

Taking into account the above requirements, Table 1 classifies the currently available VPN options. The list of VPNs included in the table has been constructed based on web searches and GitHub repositories providing an overview of currently available VPN options.[30,31] Compliance with the outlined features was determined by browsing the official documentation pages, reading the source code, as well as hands-on testing.

Out of the 11 considered options, 4 satisfy the essential requirements discussed above: Headscale, Netbird, ZeroTier, and Netmaker. Internet and Nebula satisfy all but one requirement, that of providing relaying over TCP in cases where direct connectivity is not possible between the nodes. Such obstacles can occur when devices are placed behind restrictive firewalls which completely prevent either outgoing, incoming or both outgoing and incoming UDP traffic. While we do recognize that TCP relaying is suboptimal, in certain cases it is better to have at least some form of connectivity, even though with lower throughput and potentially increased latency, than having no connectivity whatsoever. Interestingly, all considered options support both the x86 and ARM architectures. It should be pointed out that Headscale and Tailscale share the same client applications, but Headscale is an open-source implementation of the otherwise closed source control plane of Tailscale.

Looking at the optional features, the four selected options all support-designating devices as subnet routers for the purpose of accessing other hosts part of the same local network, but not connected directly to the VPN mesh. Headscale also supports granular ACLs with rules that are written using a JSON-like syntax, and synced via the control plane to the local clients running on the nodes taking part in the mesh.[32,33] Using such ACL rules it is possible to divide the nodes into groups and explicitly define on what ports they are allowed to communicate. It should be noted that these traffic rules are in addition to any host firewall rules that the user might have configured themselves on the nodes and are made possible by the Tailscale client itself and the Wireguard implementation running in user-space.

TABLE 1 Comparison of VPN solutions and their classification according to the outlined criteria.

	General information			Criteria compliance						
Name	**Commits[a]**	**Protocol**	**Stars[b]**	**OS**	**Recent release[c]**	**Mesh[d]**	**UR**	**ACL**	**Arch**	**ACO**
Headscale[34]	2941	Wireguard (US)	13.6 k	✓	✓	✓	✓	✓	x86 & ARM	✓
Netbird[35]	903	Wireguard (US & KS)	4.6 k	✓	✓	✓	✓	✓	x86 & ARM	✓
ZeroTier[36]	6108	Custom	11.4 k	✓	✓	✓	✓	✓	x86 & ARM	✓
Netmaker[37]	5292	Wireguard (US & KS)	6.9 k	✓	✓	✓	✓	✓	x86 & ARM	✓
Tailscale[38]	5802	Wireguard (US)	12.4 k	X[e]	✓	✓	✓	✓	x86 & ARM	✓
Firezone[39]	2212	Wireguard (US & KS)	4.3 k	✓	✓	X	X	✓	x86 & ARM	X
WgEasy[40]	141	Wireguard (US & KS)	7.5 k	✓	✓	X	X	X	x86 & ARM	X
Mistborn[41]	220	Wireguard (US & KS)	583	✓	X	X	✓	✓	x86 & ARM	X
Wesher[42]	186	Wireguard (KS)	802	✓	X	✓	✓	X	x86 & ARM	X
Innernet[43]	322	Wireguard (KS)	4.3 k	✓	✓	✓	✓	✓	x86 & ARM	X
Nebula[44]	383	Custom	11.9 k	✓	✓	✓	✓	✓	x86 & ARM	X

Abbreviations: ACO, advanced connectivity options, including traffic relaying over both TCP and UDP; Arch: architecture support; KS, kernel-space; OS, open-source; UR, unattended registration; US, user-space.
[a]Number of GitHub commits as of 2023-06-17.
[b]Number of times the Git repository has been starred or added to favorites (depending on the Git service used) as of 2023-06-17. Used as a popularity metric.
[c]Release in the last six months, counting from 2023-06-17.
[d]Full-mesh support.
[e]The client side applications are open-sourced while the control plane is not. Headscale is an open-source implementation of the Tailscale's proprietary control plane.

TABLE 2 Details about the testing environment.

Component	Description
CPU	Intel Xeon x5647, 2.93GHz, 4 cores
Storage	320GB mechanical hard drive
Memory	8GB DDR3
Operating system	Ubuntu 22.04 LTS
Kubernetes version	1.22.17
K3s version	v1.22.17+k3s1
Headscale version	v0.21.0
Tailscale client version	v1.42.0
Netbird version	v0.20.0
ZeroTier version	v1.10.6
Netmaker version	v0.20.0

3.2 | Testing environment setup

All of the benchmarks were conducted on a dedicated testbed comprised of six identical bare-metal nodes. The characteristics for each of the nodes are given in Table 2. Each node had a distinct role, as follows:

- 1 node acting as a dedicated Kubernetes master node, hosting only the core Kubernetes and CNI components, and no user provisioned workloads.

- 2 nodes acting as Kubernetes worker nodes. In order to guarantee the accuracy of the results, no other workloads were scheduled on these nodes apart from the required containers for the Calico CNI plugin and the containers in which the benchmarks themselves were executed.
- 1 node for deploying the relaying component of the various VPN solutions.
- 1 node acting as a firewall and a router with PfSense[45] installed, allowing full control over the communication between the nodes, and simulations of different scenarios in terms of connectivity restrictions.
- 1 auxiliary node acting as a hypervisor, allowing the creation of VMs for hosting additional components which were neither performance nor throughput sensitive, required for some of the tests.

All of the nodes were connected to the same 1 Gbit/s network switch and each placed in their own isolated VLAN, with the PfSense firewall acting as the default gateway and routing traffic between them. Placing the Kubernetes nodes in isolated layer 3 domains allowed us to simulate a distributed environment, where the different nodes are not part of the same local network. The K3s Kubernetes distribution has been used for all Kubernetes deployments during the tests due to its lightweight nature, high performance, and simplicity of operation.[46] An additional bare-metal node was configured as a hypervisor, allowing us to create virtual machines for hosting some of the control plane components of the various tested solutions. It should be noted that such VMs were used only in cases where the deployed software was solely dealing with control plane traffic, and was not actively taking part in traffic forwarding. Attention was paid all traffic forwarding to be exclusively performed by bare-metal nodes directly connected to the hardware switch, ensuring the highest level of performance, eliminating potential software overheads.

Calico CNI was used as the CNI plugin in all tests. The selected encapsulation method was VXLAN. Calico itself supports two traffic encapsulation methods, IP-IP and VXLAN,[47] but we have opted for the latter since IP-IP does not support IPv6 traffic, and going forward this will become more relevant as more and more networks adopt IPv6.[48] While IP-IP has slightly smaller overhead than VXLAN encapsulation, the obtained results are still relevant, since the exact configuration was used across all test runs. When configuring the maximum transmission unit (MTU) for the CNI interfaces, care was taken to follow best practices and to account for the additional VXLAN header. More details regarding MTU configuration are available in Section 3.2.1.

Taking into account the vastly different architectures and deployment options available for each of the tested VPN solutions, we discuss specifics in dedicated subsections for each software. We strove to keep the default configuration where possible and refrain from manual source code changes, but unfortunately this was not always possible as some cases required certain manual adjustments. Even though minor, such modifications are also discussed in details in the dedicated subsections that follow since they might have an impact in real-world usage scenarios.

3.2.1 | Establishing a baseline

In order to obtain a relevant reference point to which other results could be compared to, we have established a baseline by conducting all of the defined tests without a VPN underlay network, and using MTU of 1450 bytes and an MTU of 1230 bytes. The 1450B MTU was chosen because that is the default Calico CNI MTU when it is deployed in its VXLAN mode over Ethernet.[49] We also chose to repeat the same non-VPN tests with an MTU of 1230B since this is the MTU used when benchmarking the various VPN solutions. All of the tested software, with the exception of ZeroTier, have a default MTU of 1280B on their Wireguard interfaces, and accounting for the VXLAN overhead, the recommended MTU on the Calico interface itself, when there is such a VPN underlay is 1230B. In the case of ZeroTier, the default MTU is 2800B, but this can easily be adjusted manually by the operator.

3.2.2 | Headscale

Headscale is the name of the open-source control plane implementation compatible with the Tailscale Software as a Service (SaaS) client applications. Although not affiliated with the Tailscale company, it provides the possibility to have a fully self-hosted VPN solution, without relying on the otherwise cloud based control-plane managed by Tailscale. Since the official Tailscale client applications are already open-source and can be used to connect with third-party control plane

implementations, they can be reused in such self-hosted scenarios as well. In the rest of the text we will refer to this VPN solution simply as Headscale.

Headscale uses the Wireguard user space implementation and supports the definition of granular ACL rules which can be used in addition to an already configured network or host based firewall solution. These ACLs, in the case of the self-hosted version, are defined in a JSON like file on the node where the Headscale control plane is deployed and are then distributed to all the nodes to which they apply. Only devices which have an explicit rule allowing traffic between them form a connection, leading to the possibility for multiple independent and mutually isolated mesh network to coexist and be managed by the same control plane. The default MTU on the automatically configured Wireguard interface on the client devices is 1280B. A web interface is also provided which can be integrated with third-party Identity Providers (IdP), allowing easy device addition and removal.

To facilitate connections in networks that are difficult to access, Tailscale has defined designated encrypted relay for packet (DERP) servers.[50] These servers are capable of relaying the VPN traffic between nodes over TCP on port 443, masquerading it as HTTPS traffic which is commonly allowed on even restrictive networks. In case a direct connection is possible between the nodes, then, once full connectivity is established, such an intermediary relaying solution is no longer used. Today there are around a dozen globally distributed DERP servers hosted by Tailscale available for free use, but there is also a possibility to self-host a DERP server. In our case, since the aim is to have complete control over all aspects of the VPN solution, we have opted to locally self-host such a DERP server on one of the bare-metal nodes, which is possible by compiling the DERP server source code.[51] Headscale can then be reconfigured to only advertise the self-hosted DERP server, instead of the globally available alternatives managed by Tailscale.

The use of an intermediary DERP server for relaying is transparent to the user from the configuration perspective, since the Tailscale client software is capable of automatically detecting whether direct connection is possible or not and mitigating the situation accordingly. Of course, when DERP relaying is in effect the latency will be higher and the throughput lower, but this can be overcome to an extent by self-hosting a DERP server in a strategically chosen geographic location with sufficient network capacity. Switching from a DERP relayed connection to a direct one or vice versa is near instantaneous with no prolonged connectivity loss.

3.2.3 | Netbird

Netbird is an open-source software which uses the Wireguard protocol for establishing full-mesh encrypted tunnels between devices. It supports both the user space and kernel space implementations of Wireguard, depending on the operating system and hardware where it is installed. A web interface for the administration of the control plane is also available for self-hosting, which similarly to Headscale, can be integrated with an IdP for easier device lifecycle management. The default MTU on the Wireguard network interface is also 1280B. Despite the fact that it uses Wireguard under the hood, it comes with a custom open-sourced client which needs to be installed on participating devices to facilitate communication with the central control plane.

Netbird advertises relaying support in cases where no direct connection is possible between the participating devices in the mesh. The traffic relaying uses the established and well-known TURN protocol.[52] The recommended TURN server is Coturn.[53] During the software selection phase, we looked at the documentation page of Netbird and at the time there was no explicit mention whether TURN relaying is supported over both UDP and TCP. Knowing that the Coturn software supports both, and finding references to relaying protocol selection in the Netbird source code,[54] we assumed that indeed it is the case that both transport protocols are supported. During the testing phase, we have confirmed that TURN relaying over UDP is working, but it seems that support for TCP relaying is not yet fully implemented, and as of the time of writing it is not a supported use-case. Despite this, we chose to include Netbird and to compare its direct and relaying performance with the other solutions, since it uses kernel space Wireguard which is a popular choice today, albeit not supported by either Headscale or ZeroTier. We deemed it interesting to see how the user space implementation of Headscale would compare to the Netbird kernel space.

Since in our testbed the transition between UDP TURN relaying and direct connection was not reliable, we chose to manually apply a patch which has not yet been merged to the master branch at the time, in order to force connection relaying and recompile the netclient software running on the end-devices.[55] Another configuration change which needed to be performed to ensure the reliable operation of the VPN once an overlay network was deployed over it using Calico was to blacklist the Calico interfaces so that they are not used for VPN communication between the participating devices. This solves the problem of circular dependency, where the Calico overlay network is deployed over the VPN, but the VPN

software sees the newly available interface as another path to reach the other peers, and chooses that one over the default, hardware based, network interface. In such cases, where the virtual interface is preferred, the VPN communication is torn down, and the mesh network is no longer functional. As was the case with the DERP relay for Headscale, the Coturn TURN server was hosted on a dedicated bare-metal node in the testbed.

As of now there is support for coarse ACLs, allowing or denying traffic between nodes without more granular control at the port level, which is sufficient for using the same control plane for hosting multiple independent mesh device networks.

3.2.4 | ZeroTier

ZeroTier is also a SaaS product, similarly to the case with Tailscale, which has also been open-sourced to an extent. It is possible to run a self-hosted control plane, albeit without an official web interface. There is only an officially provided REST API for managing networks and devices, although there are third-party projects which can be used to also deploy a web interface, if needed.

ZeroTier does not use an existing VPN protocol, but implements a custom one instead.[56] Its architecture, at first glance, is somewhat more complicated than the rest of the options introduced so far. ZeroTier mesh networks are comprised of end-devices, network controllers and roots. End-devices run the zerotier-one client software through which they can join and leave existing ZeroTier networks. Network controllers are responsible for member addition, certificate management, and configuration syncing between participating members in the mesh. Finally, roots are responsible for discovery and connection establishment between participating nodes, and can also relay traffic when no direct peer-to-peer connection is possible between mesh nodes. Both network controllers and roots are hosted by ZeroTier, with an option to also self-host them on private infrastructure. While hosting a custom network controller is a well-documented and straightforward process,[57] hosting a custom root is not. The use of a custom root server requires the creation of a custom "world" file, which requires manual source-code changes, as well as client configuration changes.[58]

Traffic relaying over TCP is also supported, and there is a network of globally distributed TCP relays managed by ZeroTier and reachable via any cast. However, during our testing these relays were unreachable, and, as discussed previously, our end-goal was to self-host this functionality anyhow. To do so, a TCP proxy implementation is open-sourced, which can be compiled manually using its source code.[59] Once up and running, configuration changes need to be done to all ZeroTier client nodes so that they are made aware of the TCP proxy to use. The TCP proxy was hosted on a dedicated bare-metal node.

As in the case of Netbird, to guarantee reliable connectivity in scenarios when an overlay network is established over the VPN interfaces, the Calico virtual interfaces have to be explicitly blacklisted in the client configuration files.

When it comes to device isolation and ACLs, ZeroTier is reminiscent of more traditional, physical, networks. A given device can be part of multiple ZeroTier networks at the same time, and in cases multiple meshes need to be constructed using the same control plane, this would translate to the definition of multiple ZeroTier networks, each containing unique client devices.

3.2.5 | Netmaker

Netmaker is also an open-source full-mesh VPN solution which is based on the Wireguard protocol. It is very similar to Netbird since it also supports both the user space and kernel space implementations of Wireguard, depending on the environment where it is installed. Netmaker has only recently introduced traffic relaying support for interconnecting devices where no peer-to-peer connection is possible. The proposed solution is also based on the TURN protocol, same as with Netbird. We have evaluated the first version (v0.20.0) of Netmaker that supports TURN relaying, and unfortunately we have encountered issues which made it impossible to include it in the results. The process for selecting whether a direct connection is possible or not was not reliable, and Netmaker was falling back to TURN relaying, even though full direct connectivity over both TCP and UDP was possible between the client devices. Additionally, even when relaying was in place, the connection was not stable and would frequently suffer downtimes in the range of 10 s of seconds, making the benchmarking process impossible. TURN relaying, in its current form, is only possible via UDP.

Taking into account that TURN relaying support was only recently introduced to Netmaker, we are confident that in the future the user experience will improve. Due to the problems that we have experienced during testing, we have decided not to include Netmaker in the list of evaluated software. It should be noted though, that Netmaker's architecture

is very similar to that of Netbird, since they both support the same Wiregurd flavors, and relaying is done over TURN, so it is reasonable to expect similar results.

3.3 | Benchmarking strategy

We have developed the benchmarking strategy focusing on five different configurations, two of which were centered around establishing a baseline without using a VPN underlay and utilizing 1450B and 1230B MTUs on the Calico VXLAN interface. The remaining three were focused on evaluating the performance of the Headscale, Netbird, and ZeroTier VPN solutions. As discussed previously, Netmaker was omitted from the tests due to the problems which were faced during its evaluation. For all three VPN products, we also defined subscenarios, simulating restricted networks which aggressively filter outbound traffic or impose difficult NATs. This was done with the end goal of testing their relaying features and performance. For both Headscale and ZeroTier this was achieved by blocking all UDP traffic using the PfSense firewall between the different VLANs where the bare-metal nodes were hosted, thus forcing the fallback to relaying over TCP. For Netbird, once it was determined that TCP relaying over TURN is currently in fact not possible, we blocked all UDP traffic between those VLANs hosting the Kubernetes nodes, while allowing UDP traffic only towards the dedicated bare-metal node hosting the TURN server.

In total there were eight different benchmarking scenarios across the five different configurations, and in each a Calico overlay was deployed:

- MTU 1450B without VLAN underlay.
- MTU 1230B without VLAN underlay.
- Headscale with direct connections between the nodes.
- Headscale with relaying via TCP using a DERP server placed in an isolated VLAN, but on the same hardware switch as the rest of the nodes.
- Netbird with direct connections between the nodes.
- Netbird relaying via UDP over TURN, with a Coturn server placed in an isolated VLAN, but on the same hardware switch as the rest of the nodes.
- ZeroTier with direct connections between the nodes.
- ZeroTier relaying via TCP using the official TCP proxy placed in an isolated VLAN, but on the same hardware switch as the rest of the nodes.

For each of these eight different benchmarking scenarios, the following tests were performed:

- Kubernetes Pod-to-pod TCP and UDP throughput tests, respectively, with inter-pod communication over the Calico overlay. Each test run was executed for 60 s and repeated 100 times.
- Kubernetes Pod-to-pod TCP and UDP throughput tests, with induced packet loss of 1%, 5%, and 10% respectively on the Calico interface. Each test run was executed for 60 s and repeated 10 times.
- Kubernetes Pod-to-pod TCP and UDP throughput tests, with induced packet delay of 50, 250, and 350 ms, respectively on the Calico interface. Each test run was executed for 60 s and repeated 10 times.
- Kubernetes API response time test, with 10 iterations per scenario, where each iteration consisted of 5000 requests towards the Kubernetes API with a maximum of 10 requests per second.

During all throughput tests CPU usage was monitored as well, as to be able to evaluate the efficiency in terms of required compute performance of each of the VPN solutions. The number of execution runs between the basic TCP and UDP tests, and the ones with induced network issues differ, since the one performed under ideal conditions were used to evaluate the core characteristics of each of the tested solutions. The remaining tests were performed solely to verify the feasibility of using the VPN solution in nonideal conditions such as those with high packet loss or high packet delay. The values for the induced packet delays were chosen as to model globally distributed clusters, and in the extreme case very busy networks with inadequate Quality of Service (QoS) rules configured. Similarly, the packet losses of 1%, 5%, and 10%

in the most extreme case were chosen to represent unreliable network links which can appear due to a number of reasons in reality, such as malfunctioning equipment or high traffic levels which overwhelm the buffers of intermediary network devices. The traffic control (tc) utility was used to simulate both the packet loss and the packet delays.[60]

We have utilized an adapted version of the Kubernetes network benchmark (KNB) suite[61,62] for all pod-to-pod throughput test. KNB uses the popular and well-known iperf3 utility for conducting the tests across two pods, where one of the pod is acting as a server, and the other one as a client. Despite the fact that all of the bare-metal nodes used for the experiments have the same exact hardware configuration, in all cases we fixed the placement of the server and client iperf3 pods on the same Kubernetes worker nodes, in essence dedicating one bare-metal node to the server and another one to the client. This was done as to avoid any unforeseen inconsistencies in the results which might have occurred should the client and server have been randomly distributed across the nodes between different executions.

Each iteration of the Kubernetes API response time benchmarks was done from a pod running in the cluster which used the Hey benchmarking utility,[63] acquiring the list of running pods from the Kubernetes API. Kubernetes API response times are a very important metric, since all of the communication with the Kubernetes control plane is done via the API. Potentially inconsistent performance or prolonged unavailability of the API might jeopardize the operation of the complete Kubernetes cluster, since there would be no coordination mechanism between the participating nodes.

4 | RESULTS

We discuss the obtained results from the previously introduced benchmarks in the subsections that follow. We begin by focusing on the TCP throughput tests in Section 4.1, before moving to the scenario where UDP was used as the underlying transport protocol during the benchmarks' execution, in Section 4.2. We conclude this section with Section 4.3, where we discuss the measured Kubernetes API response times when each of the tested VPN solutions is utilized as the underlay network.

4.1 | TCP throughput

For each of the evaluated seven scenarios, we tested the TCP pod-to-pod throughput performance. Figure 1 presents the obtained results, as well as the CPU usage during each test case. Taking into account the limited space, the Appendix with Table A1 presents the mean throughput values, while also comparing them to the two baselines.

Starting the discussion with the scenario where no adverse network conditions are present and there is no simulation of either packet loss or packet delay, the best throughput, unsurprisingly, is achieved by the MTU 1450B test case, in which no VPN network underlay was utilized. Both baselines, MTU 1450B and MTU 1230B show comparable CPU usage, with MTU 1450B achieving better throughput, which can be attributed to the larger MTU size. Analyzing the results from the VPN solutions, Netbird in its direct connection mode has a clear advantage over the rest when it comes to achieved throughput, with a mean value of 707.45 Mbps, surprisingly similar to the mean value achieved by the MTU 1230B which is 709.94 Mbps. This performance can be attributed to the efficient kernel space implementation of Wireguard which was used by Netbird on the test infrastructure. The second place when it comes to overall throughput goes to ZeroTier, while Headscale's user-space Wireguard implementation is the slowest. Interestingly, looking at the CPU utilization, ZeroTier is the most resource efficient, both when a direct connection is possible, as well as when relaying traffic over the TCP proxy. The Netbird UDP relay has the highest CPU utilization, while offering worse performance than ZeroTier's relay, but better than Headscale's TCP relay.

It should be mentioned that in all test cases, the expected level of performance of the VPNs should be noticeably worse than direct connectivity without VPN, since there are multiple encapsulation layers involved; starting from the VPN tunnels, which can also be relayed over either TCP or UDP when no direct connectivity is possible between the nodes, and then the Calico overlay network which encapsulates pod traffic using VXLAN, and finally the application layer traffic, which in this case is generated by iperf3 for the purposes of the throughput tests.

Looking at the results where additional delay is gradually introduced, we notice that the MTU 1450B baseline still offers the best performance for 50, 250, and 350 ms delays. With the increase in the delay, Netbird's relaying performance starts to diminish, and achieves worst results in all three delay scenarios, which was not the case previously. On the other hand, Headscale now achieves comparable results to the rest of the solutions, and has the best mean throughput (156.3 Mbps) in the 50 ms delay test, followed by ZeroTier (155.6 Mbps). As the delay increases, all VPN solutions start

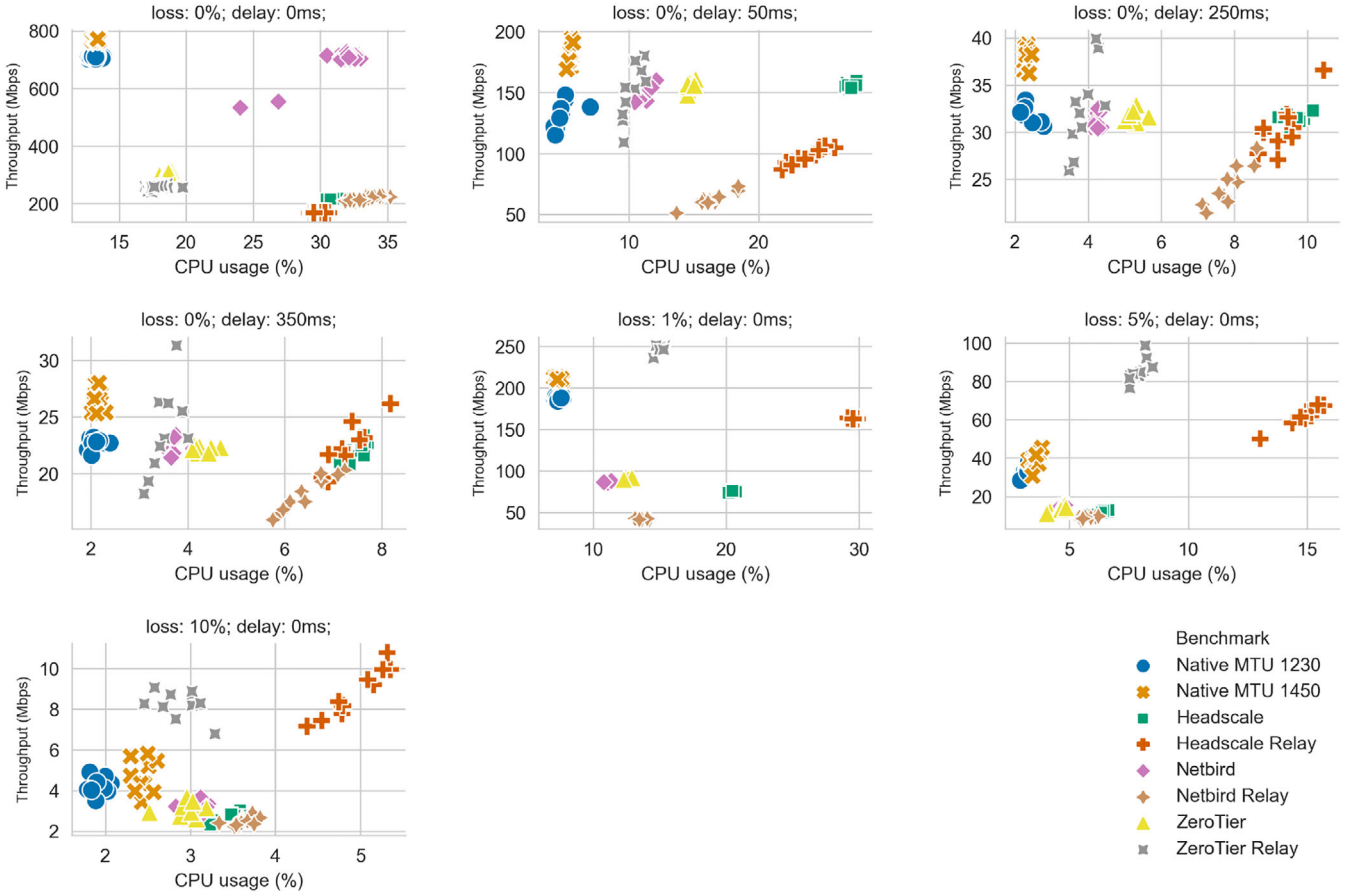

FIGURE 1 TCP throughput and CPU usage for each test scenario.

to show similar performance. In fact, not taking into account the relaying performance, in the case where 350 ms additional delay is added, all VPN solutions show an average throughput of 22 Mbps. ZeroTier offers less consistent results characterized by large standard deviation across all three delay scenarios. When it comes to relaying, we conclude that both Headscale and ZeroTier achieve better performance than Netbird's UDP relay, due to their utilization of TCP as the underlying transport protocol, offering reliable data transfer in the face of worsening network conditions. Analyzing CPU usage, ZeroTier with its custom VPN protocol still consistently uses the least resources across all three delay scenarios, with Headscale being the most taxing implementation, a fact that can also again be attributed to its user-space implementation of Wireguard.

Moving forward to the tests where packet loss has been gradually increased to simulate intermittent connectivity issues, we notice a very interesting occurrence. The relaying options of both Headscale and ZeroTier offer consistently and drastically better performance than all other counterparts, including the baselines of 1450B and 1230B MTUs, when no VPN underlay is being utilized. One thing which both the Headscale and ZeroTier relay strategies have in common is that they use TCP on the transport layer when relaying. TCP's advanced congestion control mechanisms result in almost two times better average throughput than all other alternatives. Analyzing the CPU usage, Headscale's relaying implementation shows higher CPU usage than ZeroTier's.

Going back to the direct connection performance of Netbird when no adverse network conditions are introduced and taking into account that the measured performance is very close to the 1230B MTU baseline, we performed the nonparametric Mann-Whitney *U*-test with the aim of verifying the statistical significance of the results. The following hypotheses were used together with an alpha value of 0.05:

- H0: the two populations are equal
- H1: the two populations are not equal

The obtained *p*-value is 0.39, thus showing that there is not enough evidence to reject the null hypothesis and conclude that there is a statistically significant difference in the obtained results. In all other cases the results were statistically significant with a *p*-value smaller than alpha, thus leading to the rejection of the null hypothesis.

4.2 | UDP throughput

Similarly to the previous case, Figure 2 shows the UDP throughput as well as the observed CPU usage during the test runs for the seven evaluated scenarios. Table A2 in the Appendix presents the descriptive statistics, as well as how the mean throughput compares to the two baselines when 1230B and 1450B MTUs are configured without a VPN underlay network.

Beginning the discussion with the case where no adverse network conditions have been simulated, it can unsurprisingly be seen that both baseline cases where no VPN is used show the best performance, which is exactly identical to the TCP case. Netbird still offers the highest performance, which is also consistent to the TCP case, with ZeroTier being in second place, and Headscale in third. Looking at the relaying performance, Netbird's UDP has the highest mean throughput with 289.51 Mbps, followed by Headscale's and ZeroTier's TCP implementations with 256.7 and 227.09 Mbps average throughput, respectively. Headscale has highest CPU usage, upwards of 40%, while ZeroTier, similarly to previous discussions, has the lowest.

Continuing the discussion with the case when delay is introduced, and looking at the 50 ms visualization, we can notice a drastic increase in the standard deviation for all VPN solutions except for Headscale and Netbird's UDP relay. Headscale also achieves the best throughput with a mean value identical to that of the 1230B MTU baseline (187.5 Mbps). Netbird's relay and ZeroTier closely follow with mean values of 183.7 and 193.11 Mbps, respectively. ZeroTier's relay mean throughput of only 92.76 is the lowest, hindered by the inconsistent performance and high standard deviation. However, as the delay increases, so does ZeroTier's relay performance, achieving the best result in both the 250 and 350 ms test cases, higher than even the 1230B baseline. While the rest of the results are comparable to each other, it is evident that Netbird fares worse for UDP traffic when network delays are present, achieving worse results in all three delay test cases.

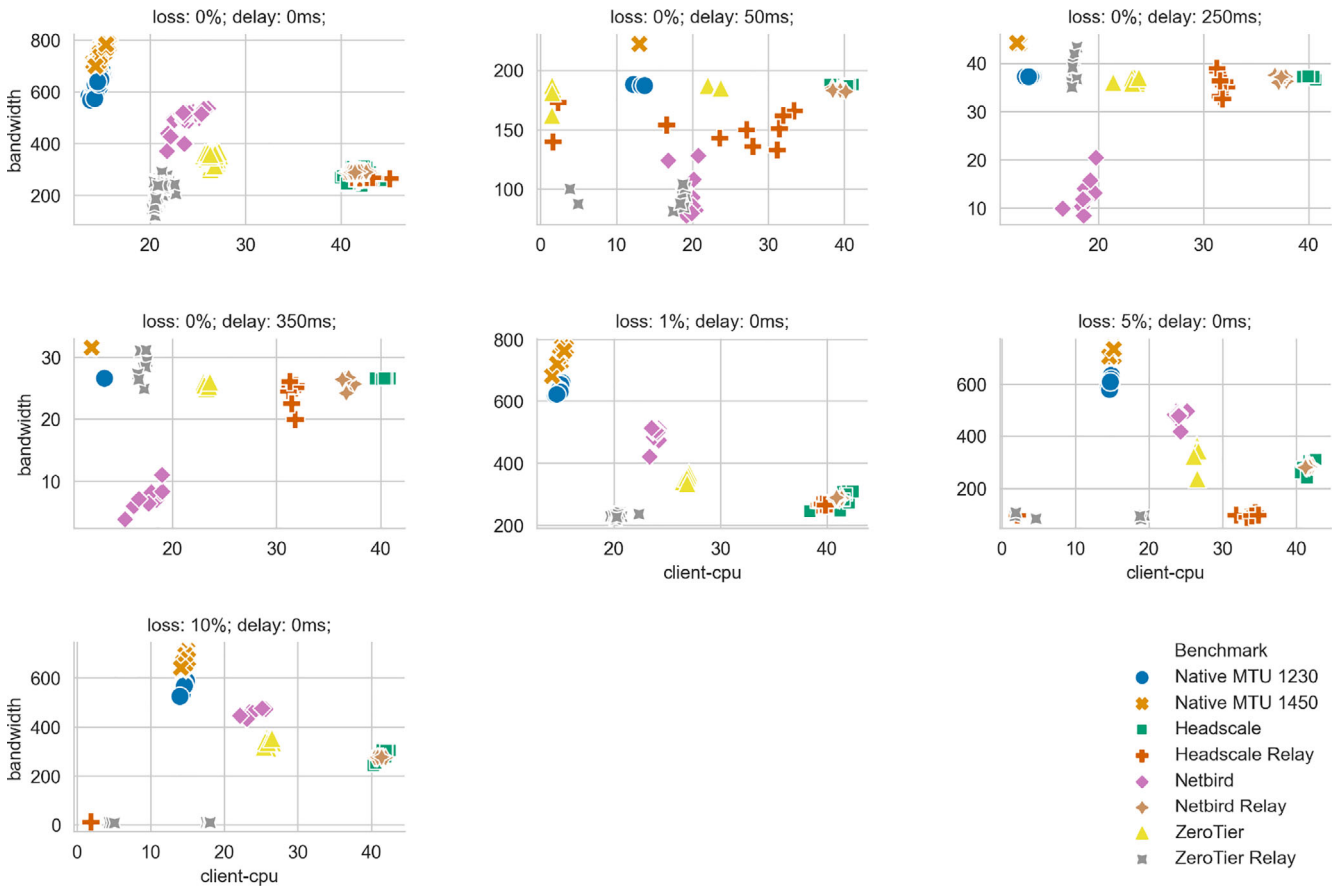

FIGURE 2 UDP throughput and CPU usage for each test scenario.

Nonetheless, Netbird's outlook significantly improves in the packet loss scenarios, where it takes the lead for 1%, 5%, and 10% packet loss. Contrary to the TCP analysis, the TCP relay options of both Headscale and ZeroTier now offer the worst performance, leading to the conclusion that TCP is not a good match for relaying UDP application layer traffic, as is the case with the iperf3 generated data.

Focusing on CPU usage, we see greater overall CPU usage for the UDP results, compared to the TCP results. ZeroTier and Netbird consistently show the lowest CPU usage of all tested solutions, excluding of course Native MTU 1230 and Native MTU 1450. This is even the case in delay 250 and delay 350, where in addition to ZeroTier having low CPU usage, it also achieves the best performance. Headscale, with its user-space implementation of Wireguard consistently shows the highest CPU usage. In certain cases relaying is more efficient in terms of CPU usage than native, but this is corresponding to cases where the relay throughput is lower than that of a direct connection.

Since the performance of Headscale between the direct connection scenario and relaying is comparable, we conducted another batch of the nonparametric Mann-Whitney *U*-test, with the following two hypotheses and an alpha value of 0.05:

- H0: the two populations are equal
- H1: the two populations are not equal

Both in the case of Headscale direct and relay, as well as in all remaining ones, the *p*-value was smaller than alpha, thus leading to the rejection of the null hypothesis, validating the statistical significance of the obtained results.

4.3 | Kubernetes API response times

The final benchmark which was performed is the Kubernetes API response time benchmark, evaluating the overall responsiveness of the Kubernetes API when different VPN solutions are used for the underlay network. Figure 3 visualized the mean response times of the Kubernetes API with a limit of a maximum 10 requests per second.

Analyzing the results, the two baselines again show the best performance, as expected. Focusing on the VPN solutions, in this case Netbird achieves the best performance, followed by ZeroTier, and then Headscale. This is exactly equivalent to the achieved performance of both the TCP and UDP throughput tests without adverse conditions. When it comes to relaying, the performance between ZeroTier and Headscale is very similar, while Netbird's UDP relaying offers worse results. To verify the statistical significance of the obtained results, we have again utilized the pair-wise nonparametric Mann-Whitney *U*-test with an alpha value of 0.05 and the hypotheses:

- H0: the two populations are equal
- H1: the two populations are not equal

Testing the results obtained from all tested solutions between each other, we can conclude that there is not enough evidence to reject the null hypothesis in only a single case, that of Native MTU 1450B and Native MTU 1230B, with a

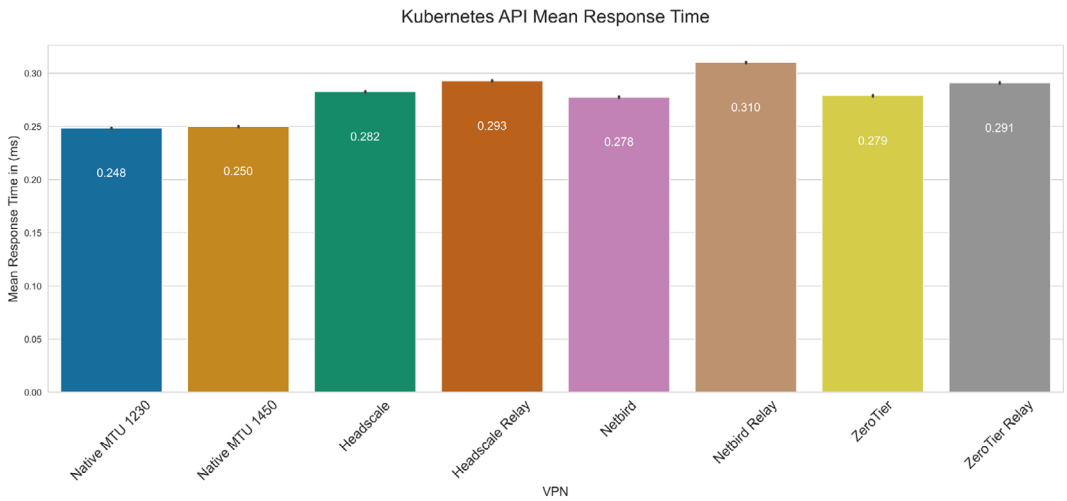

FIGURE 3 Kubernetes API mean response time for different VPN solutions.

p-value of 0.22. In all other cases the *p*-value is smaller than alpha, leading to the rejection of the null hypothesis, proving the statistical significance of the results.

5 | DISCUSSION

Looking at the obtained results, it is evident that the final choice would depend on the advantages and disadvantages that each tested solution showed, and how these relate to the given use-case at hand. In this context, attention should be paid so that raw performance is not the only metric taken into account when evaluating which VPN software to use; the on-boarding experience, the overall reliability of the established connection, the deployment complexity, are all aspects which need to be considered as well.

To aid the discussion on the strengths and weakness of the tested solutions, Figures 4 and 5 present the mean TCP and UDP throughput in Mbps achieved during the testing of the various VPN solutions.

Focusing initially on raw throughput, Netbird shows the best performance in cases where the network connection between the nodes is reliable, and there is no significant packet loss or packet delay. The kernel based Wireguard implementation which Netbird offers adds minimal overhead compared to scenarios when no VPN is present. However, when it comes to scenarios where the communication latency might be high between a set of nodes, both Headscale and ZeroTier offer better performance. They are closely matched both for TCP and UDP traffic, so in such cases the final choice will need to be made based on the other aspects inherent to each solution, such as the on-boarding experience, discussed in more details below. Concluding the throughput analysis with the last set of results focused on unreliable network connectivity between the nodes when packet loss regularly occurs, it can be seen from Figure 4 that ZeroTier offers the best performance in all but one case, that of 10% packet loss, where Headscale takes the lead. Interestingly, another conclusion can also be drawn from the same results. It is more preferable to leverage the TCP relaying functionality of either Headscale or ZeroTier and forward traffic via a third party node in such cases, with the aim of achieving higher throughput and potentially avoiding the network segment where packet loss occurs. These conclusions do not apply when the upper layer protocols use UDP for transport, since in those cases under lossy conditions, Netbird takes the lead again and achieves the best performance.

Moving on to the remaining aspects in addition to throughput, when it comes to the effort required for self-hosting all of the necessary components, both Headscale and Netbird have a clear advantage over ZeroTier. ZeroTier discourages the hosting of individual roots which play a role in the control plane, and doing so requires manual changes to the software. While the process is documented, it is also time consuming and requires setting up a development environment to make the necessary adjustments. The ZeroTier mobile applications also do not support the use of custom roots as of the time of this writing,[64] while the desktop ones do via the discussed manual changes in Section 3.2.4. ZeroTier also has an additional

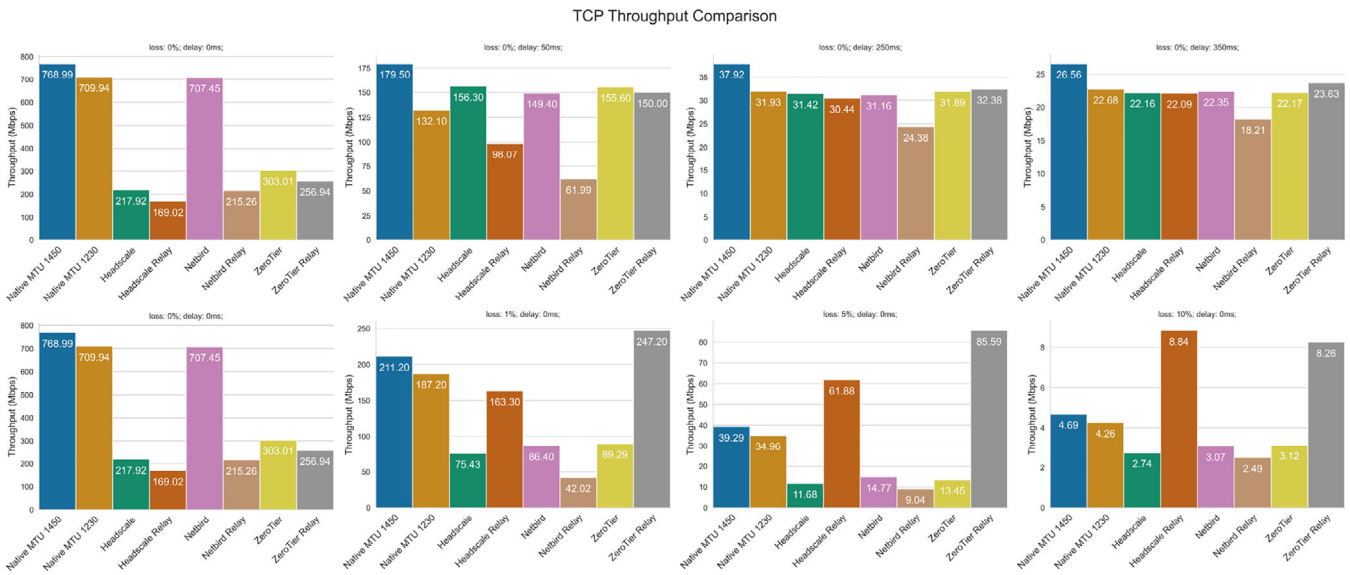

FIGURE 4 TCP mean throughput in Mbps.

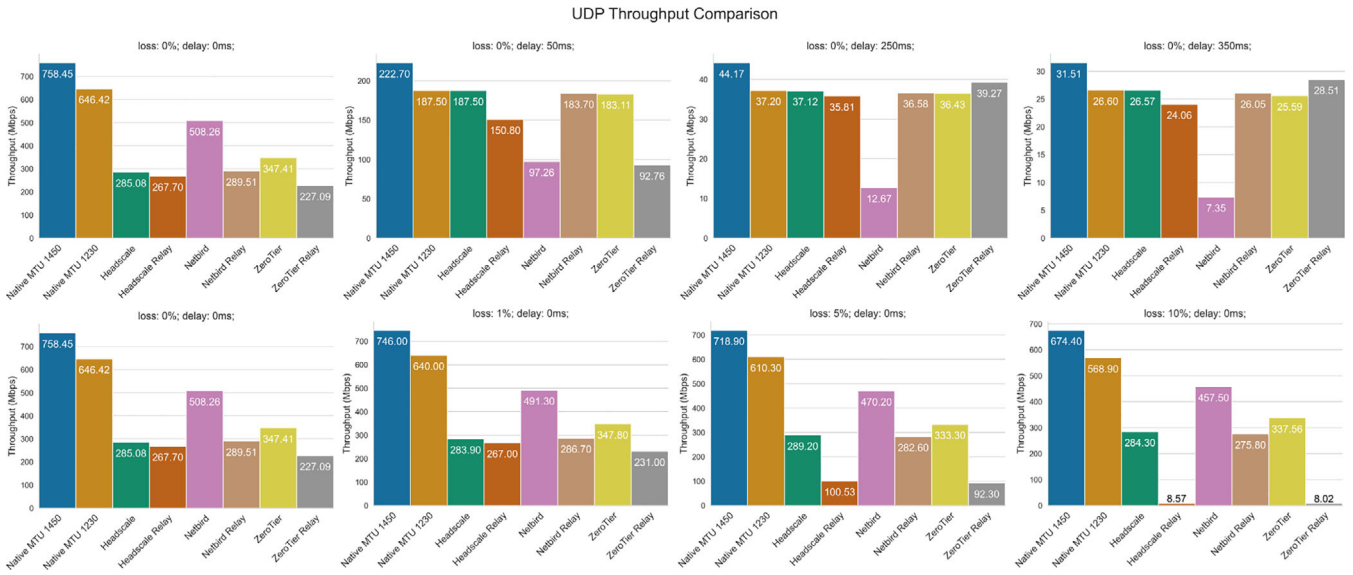

FIGURE 5 UDP mean throughput in Mbps.

drawback, which is the very high convergence time when relaying needs to be put in place. During the tests we have measured this time to be in the range of 3–4 min. Such issues were not present in either Headscale's or Netbird's case.

In the case of Netbird, as of now, only TURN relaying over UDP is supported, but we hope to see TCP support implemented as well in the future. During the testing and with the latest available version of Netbird at the time, the fallback mechanism to relaying was in certain cases unreliable, and we had to implement source code changes to always fall back to TURN relaying in order to be able to test this scenario.

Headscale, as a self-hosted version of the Tailscale control plane, did not present any major issues during our testing. Additionally, Headscale also supports granular ACLs which are enforced by the Tailscale client running on all of the nodes. One drawback which is currently present is that updating the ACLs needs to be done in a JSON file on the server where Headscale itself is hosted, but for the changes to take effect, the whole control plane needs to be restarted. No gradual reload is supported as of the time of this writing, a feature which is available in the hosted version.

Finally, when it comes to user experience, only Netbird offers an officially sanctioned, built-in, web interface. As for the rest, both Headscale and ZeroTier offer REST APIs in the self-hosted versions, but third-party projects are available which can build a graphical user interface around them.

6 | CONCLUSION

The introduction of the Wireguard VPN protocol has ushered a new era of VPN solutions, and has made it feasible to construct full-mesh topologies between VPN clients as a result of its low resource usage. However, due to the inherent scalability problems with manual configuration of a large number of nodes, automated solutions in the form of centralized control planes were quickly developed. These control planes facilitate the process of node discovery and key exchange for the machines taking part in the mesh network. Such dispersed full-mesh topologies made up of globally distributed compute servers have an incredible potential to improve the coordination between the edge and the cloud, introducing a cloud-edge continuum where even sensitive data can securely and reliably be moved between the environments.

In order to evaluate the current VPN software possibilities in this space, we have constructed a set of criteria that prospective VPN solutions need to meet before they can be used as an underlay network on top of which other overlay networks can be deployed, for example as a part of a more complex deployment of orchestration solutions such as Kubernetes. By applying the criteria which includes both essential and optional features, we have selected three self-hosted, full-mesh VPN options for evaluation. Through a set of benchmarks consisting of pod-to-pod TCP throughput, pod-to-pod UDP throughput and Kubernetes API server response time, we have evaluated the Headscale, Netbird, and ZeroTier solutions, as well as their robustness when faced with intermittent connectivity issues, such as large packet delay or packet loss.

Analyzing the results shows that the final choice of the VPN solution depends on the overall requirements and network conditions. Based on this, Netbird, with its Wireguard kernel implementation offers near native performance assuming

the network environment is stable, without large packet loss or packet delay. Netbird is also the recommended solution in scenarios where packet loss is expected, but only when the majority of the traffic being tunneled is using the UDP transport protocol. This is not the case when tunneling TCP traffic though, since both Headscale and ZeroTier show dramatically better performance in such circumstances. Finally, when it comes to packet delays, Headscale and ZeroTier also offer better performance compared to Netbird.

Focusing on the usability aspects, Headscale offers a more optimal self-hosting user experience, requiring no manual changes to the source code to achieve a fully self-hosted control plane that is robust enough to graciously fall back to TCP relaying when direct connectivity is not possible between nodes. Headscale also supports granular ACLs which can further increase the overall security of the network, in tandem with traditional host and network based firewalls. However, its user space implementation of Wireguard offers noticeably lower performance than the other tested solutions, so it is a compromise between ease-of-use and performance.

In conclusion, current VPN options are capable of establishing a full-mesh network which can be reliably used for setting up distributed Kubernetes clusters on top of it. The relaying functionality present in all three solutions, guarantees that nodes can be placed even in networks with complex NAT setups or restricted connectivity which prevents peer-to-peer communication between the nodes. We expect that the full-mesh VPN landscape will continue to improve, both with brand new self-hosted solutions, as well as additional improvements to existing options.

Having verified the applicability of VPN solutions for constructing an underlay network which can be used to set up an overlay using some of the various Kubernetes CNI plugins, our plans for the future are to develop a comprehensive Kubernetes federation strategy, where multiple clusters, placed both in the cloud and in the edge, can be interconnected with each other and can be centrally managed by a single Kubernetes control plane.

7 | THREATS TO VALIDITY

The described and applied benchmarking strategy for evaluating the VPN solutions has been developed with the aim to eliminate as many threats to the validity of the results as possible. All of the servers involved in the benchmarking process have had identical hardware configuration, including the make and model of all components. Additionally, to guard against any potential inconsistencies due to network factors not directly under our control, such as utilization of intermediary network devices, we have decided to connect all equipment to a single 1Gbit/s network switch dedicated to the benchmarking experiments. The firewall/router device whose task was to route between the isolated VLANs utilized for the benchmarks was also dedicated solely to the experiments, and was not dealing with any other traffic or running additional services not related to the testing at hand. However, it needs to be recognized that the obtained results are relevant for the software versions which were available at the time of writing. It is expected that in the future, as new improvements are added to the various VPN solutions, new benchmarks will need to be conducted. Accounting for this scenario, we have open-sourced all of the supporting material, including the benchmarking scripts, with the hope of easing such a process in the future.

AUTHOR CONTRIBUTIONS

Conceptualization: S.F., V.K., K.G., C.B. *Investigation*: V.K. and C.B. *Methodology*: S.F., V.K., K.G., C.B. *Software*: V.K. and CB. *Validation*: S.F., V.K., K.G., C.B. *Formal analysis*: S.F., V.K., K.G., C.B. *Visualizations*: S.F., V.K., K.G., C.B. *Writing (original draft preparation)*: V.K. and C.B. *Writing (review and editing)*: S.F., V.K., K.G., C.B. All authors have reviewed the manuscript.

FUNDING INFORMATION

This study was funded by the Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, North Macedonia

CONFLICT OF INTEREST STATEMENT

The authors have no relevant financial or nonfinancial interests to disclose.

DATA AVAILABILITY STATEMENT

The generated raw data, software, and outputs from the data analysis is publicly available under a permissive license on https://github.com/korvoj/vpn-performance.

ORCID

Vojdan Kjorveziroski https://orcid.org/0000-0003-0419-4300
Cristina Bernad https://orcid.org/0000-0001-9537-415X
Katja Gilly https://orcid.org/0000-0002-8985-0639
Sonja Filiposka https://orcid.org/0000-0003-0034-2855

REFERENCES

1. Zhipeng Z, Chandel S, Jingyao S, Shilin Y, Yunnan Y, Jingji Z. VPN: A Boon or Trap?: A Comparative Study of MPLS, IPSec, and SSL Virtual Private Networks. *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE;2018:510-515. doi:10.1109/ICCMC.2018.8487653
2. Harkanson R, Kim Y, Jo J-Y, Pham K. Effects of TCP transfer buffers and congestion avoidance algorithms on the end-to-end throughput of TCP-over-TCP tunnels. *16th International Conference on Information Technology-New Generations (ITNG 2019)*. Vol 800, IEEE CCWC; 2019:401-408. doi:10.1007/978-3-030-14070-0_55
3. Mackey S, Mihov I, Nosenko A, Vega F, Cheng Y. A performance comparison of WireGuard and OpenVPN. *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. ACM; 2020:162-164. doi:10.1145/3374664.3379532
4. Wei X, Miao W, Zeng Z, et al. Research on using dynamic thread pool to improve the performance of VPN gateway. *2022 7th International Conference on Computer and Communication Systems (ICCCS)*. IEEE; 2022:566-570. doi:10.1109/ICCCS55155.2022.9846591
5. Donenfeld J. Wireguard kernel module–Linux kernel source tree. Accessed June 25, 2023. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=e7096c131e5161fa3b8e52a650d7719d2857adfd
6. Whited J, Tucker J. Userspace isn't slow, some kernel interfaces are! 2022. Accessed June 25, 2023. https://tailscale.com/blog/throughput-improvements/
7. Source code for OpenVPN community edition. Accessed June 25, 2023. https://openvpn.net/source-code/
8. Donenfeld JA. WireGuard: Next Generation Kernel Network Tunnel. *Proceedings 2017 Network and Distributed System Security Symposium*. Society; 2017. doi:10.14722/ndss.2017.23160
9. Setting up your own certificate authority (CA). Accessed June 25, 2023. https://openvpn.net/community-resources/setting-up-your-own-certificate-authority-ca/
10. Donenfeld JA. Quick start–WireGuard. Accessed June 25, 2023. https://www.wireguard.com/quickstart/
11. Tailscale. Tailscale. Accessed June 25, 2023. https://tailscale.com/
12. ZeroTier. Global area networking. Accessed June 25, 2023. https://www.zerotier.com/
13. NetBird. Zero configuration VPN for fast-moving teams. Accessed June 25, 2023. https://netbird.io/
14. Netmaker SaaS. Sign up to be one of the first users of our virtual networking SaaS platform. Accessed June 25, 2023. https://www.netmaker.io/beta
15. Tailscale: Site-to-Site networking. 2023. Accessed June 25, 2023. https://tailscale.com/kb/1214/site-to-site/
16. Kumar R, Trivedi MC. Networking analysis and performance comparison of kubernetes CNI plugins. *Advances in Computer, Communication and Computational Sciences. Advances in Intelligent Systems and Computing*. Springer; 2021:99-109. doi:10.1007/978-981-15-4409-5_9
17. Pohl F, Schotten HD. Secure and scalable remote access tunnels for the IIoT: an assessment of openVPN and IPsec performance. *Service-Oriented and Cloud Computing*. Lecture Notes in Computer Science. Springer International Publishing; 2017:83-90. doi:10.1007/978-3-319-67262-5_7
18. Dekker E, Spaans P. Performance comparison of VPN implementations WireGuard, strongSwan, and OpenVPN in a 1 Gbit/s environment. https://rp.os3.nl/2019-2020/p71/report.pdf
19. Goethals T, Kerkhove D, Volckaert B, Turck FD. Scalability evaluation of VPN technologies for secure container networking. *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE; 2019:1-7. doi:10.23919/CNSM46954.2019.9012673
20. Paillisse J, Barcia A, Lopez A, Rodriguez-Natal A, Maino F, Cabellos A. A control plane for WireGuard. *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE;2021:1-8. doi:10.1109/ICCCN52240.2021.9522315
21. Gunda P, Voleti SD. Performance Evaluation of Wireguard in Kubernetes Cluster. 2021 http://urn.kb.se/resolve?urn=urn:nbn:se:bth-21167
22. Wang Z, Goudarzi M, Aryal J, Buyya R. Container orchestration in edge and fog computing environments for real-time iot applications. *Computational Intelligence and Data Analytics*. Lecture Notes on Data Engineering and Communications Technologies. Springer Nature; 2023:1-21. doi:10.1007/978-981-19-3391-2_1
23. Mlynka D. IoT device management using Kubernetes. 2022 https://is.muni.cz/th/x3jnk/fi-pdflatex.pdf
24. Falcone D. Designing a scalable network overlay for Kubernetes multi-cluster topologies. laurea, Politecnico di Torino. 2021 https://webthesis.biblio.polito.it/20504/
25. Virtual Kubelet. Accessed June 25, 2023. https://virtual-kubelet.io/
26. Subratie K, Aditya S, Figueiredo RJ. EdgeVPN: Self-organizing layer-2 virtual edge networks. *Futur Gener Comput Syst*. 2023;140:104-116. doi:10.1016/j.future.2022.10.007
27. Bittencourt L, Immich R, Sakellariou R, et al. The internet of things, fog and cloud continuum: integration and challenges. *Internet Things*. 2018;3–4:134-155. doi:10.1016/j.iot.2018.09.005
28. Keranen A, Holmberg C, Rosenberg J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*. Internet Engineering Task Force (IETF); RFC8445; 2018. doi:10.17487/RFC8445

29. Junior PS, Miorandi D, Pierre G. Good shepherds care for their cattle: seamless pod migration in geo-distributed kubernetes. *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. IEEE; 2022:26-33. doi:10.1109/ICFEC54809.2022.00011
30. Chee C. Awesome WireGuard. 2023. Accessed June 25, 2023. https://github.com/cedrickchee/awesome-wireguard
31. HarvsG: compare WireGuard mesh tools. 2023. Accessed June 25, 2023. https://github.com/HarvsG/WireGuardMeshes
32. Tailscale: network access controls (ACLs). 2023. Accessed June 25, 2023. https://tailscale.com/kb/1018/acls/
33. Font J. Headscale ACL support. 2023. Accessed June 25, 2023. https://github.com/juanfont/headscale
34. Juanfont/headscale: an open source, self-hosted implementation of the tailscale control server. Accessed June 25, 2023. https://github.com/juanfont/headscale
35. Netbirdio/netbird: connect your devices into a single secure private WireGuard®-based mesh network with SSO/MFA and simple access controls. Accessed June 25, 2023. https://github.com/netbirdio/netbird
36. Zerotier/ZeroTierOne: a smart ethernet switch for earth. Accessed June 25, 2023. https://github.com/zerotier/ZeroTierOne
37. Gravitl/netmaker: netmaker makes networks with WireGuard. Netmaker automates fast, secure, and distributed virtual networks. Accessed June 25, 2023. https://github.com/gravitl/netmaker
38. Tailscale on GitHub. 2023. Accessed June 25, 2023. https://github.com/tailscale/tailscale
39. Firezone/Firezone: WireGuard®-based VPN server and firewall. Accessed June 25, 2023. https://github.com/firezone/firezone
40. WireGuard easy. 2023. Accessed June 25, 2023. https://github.com/wg-easy/wg-easy
41. Stormblest/Mistborn GitLab. 2022. Accessed June 25, 2023. https://gitlab.com/cyber5k/mistborn
42. Antunes L. Wesher–Wireguard overlay mesh network manager. 2023. Accessed June 25, 2023. https://github.com/costela/wesher
43. *Innernet - A Private Network System That Uses WireGuard under the Hood*. Tonari, Inc; 2023. Accessed June 25, 2023. https://github.com/tonarino/innernet
44. Slackhq/Nebula. 2023. Accessed June 25, 2023. https://github.com/slackhq/nebula
45. pfSense® - World's most trusted open source firewall. Accessed June 25, 2023. https://www.pfsense.org/
46. Kjorveziroski V, Filiposka S. Kubernetes distributions for the edge: serverless performance evaluation. *J Supercomput*. 2022;78(11):13728-13755. doi:10.1007/s11227-022-04430-6
47. Overlay networking—Calico documentation. Accessed June 25, 2023. https://docs.tigera.io/calico/latest/networking/configuring/vxlan-ipip
48. Kjorveziroski V, Mishev A, Filiposka S. Evaluating IPv6 support in kubernetes. *2021 29th Telecommunications Forum (TELFOR)*. IEEE; 2021:1-4. doi:10.1109/TELFOR52709.2021.9653276
49. Configure MTU to maximize network performance—Calico documentation. Accessed June 25, 2023. https://docs.tigera.io/calico/latest/networking/configuring/mtu
50. Tailscale: DERP servers. 2022. Accessed June 25, 2023. https://tailscale.com/kb/1232/derp-servers/
51. Tailscale: custom DERP servers. 2023. Accessed June 25, 2023. https://tailscale.com/kb/1118/custom-derp-servers/
52. Mahy R, Matthews P, Rosenberg J. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. Internet Engineering Task Force (IETF); RFC5766; 2010. doi:10.17487/rfc5766
53. Coturn TURN Server. coturn. 2023. Accessed June 25, 2023. https://github.com/coturn/coturn
54. Netbird/Infrastructure_files/Management.Json.Tmpl at B524a9d49d001564b5818abe426be9689aa56ff3 · Netbirdio/Netbird. Accessed June 25, 2023. https://github.com/netbirdio/netbird/blob/b524a9d49d001564b5818abe426be9689aa56ff3/infrastructure_files/management.json.tmpl#L4
55. Add force relay conn env var for debug purpose by pappz· Pull Request #904 · Netbirdio/Netbird. Accessed June 25, 2023. https://github.com/netbirdio/netbird/pull/904
56. Protocol design whitepaper—ZeroTier documentation. Accessed June 25, 2023. https://docs.zerotier.com/zerotier/manual/
57. Network controllers—ZeroTier documentation. Accessed June 25, 2023. https://docs.zerotier.com/self-hosting/network-controllers
58. Private Root Servers—ZeroTier Documentation. Accessed June 25, 2023. https://docs.zerotier.com/zerotier/moons
59. ZeroTierOne/Tcp-Proxy at Dev · Zerotier/ZeroTierOne. Accessed June 25, 2023. https://github.com/zerotier/ZeroTierOne
60. Tc(8)–Linux manual page. Accessed June 25, 2023. https://man7.org/linux/man-pages/man8/tc.8.html
61. Lombardo F, Salsano S, Abdelsalam A, Bernier D, Filsfils C. Extending kubernetes networking to make use of segment routing over IPv6 (SRv6). http://arxiv.org/abs/2301.01178 2023.
62. K8s-Bench-Suite. infraBuilder. 2023. Accessed June 25, 2023. https://github.com/InfraBuilder/k8s-bench-suite
63. Dogan J. Rakyll/Hey. 2021. Accessed June 25, 2023. https://github.com/rakyll/hey
64. Introduction—ZeroTier documentation. Accessed June 25, 2023. https://docs.zerotier.com/self-hosting/introduction/

How to cite this article: Kjorveziroski V, Bernad C, Gilly K, Filiposka S. Full-mesh VPN performance evaluation for a secure edge-cloud continuum. *Softw: Pract Exper*. 2024;1-22. doi: 10.1002/spe.3329

APPENDIX A . DETAILED COMPARISON OF TCP AND UDP THROUGHPUT RESULTS

TABLE A1 TCP throughput results.

Test	Cond	Mean	Std. Dev.	Coef. V.	Cmp. 1450B[a]	Cmp. 1230B[b]
Headscale direct	Native	217.92	2.49	0.01	x3.53	x3.26
	L: 1%	75.43	1.10	0.01	x2.80	x2.48
	L: 5%	11.68	0.67	0.06	x3.36	x2.99
	L: 10%	2.74	0.20	0.07	x1.71	x1.56
	D: 50 ms	156.30	2.45	0.02	x1.15	x0.85
	D: 250 ms	31.42	0.68	0.02	x1.21	x1.02
	D: 350 ms	22.16	0.83	0.04	x1.20	x1.02
Headscale relay	Native	169.02	0.7	0.00	x4.55	x4.20
	L: 1%	163.30	0.67	0.00	x1.29	x1.15
	L: 5%	61.88	5.07	0.08	x0.63	x0.56
	L: 10%	8.84	1.22	0.14	x0.53	x0.48
	D: 50 ms	98.07	6.62	0.07	x1.83	x1.35
	D: 250 ms	30.44	2.64	0.09	x1.25	x1.05
	D: 350 ms	22.09	2.27	0.1	x1.2	x1.03
Netbird direct	Native	707.45	24.16	0.03	x1.09	x1.00
	L: 1%	86.40	0.72	0.01	x2.44	x2.17
	L: 5%	14.77	1.1	0.07	x2.66	x2.37
	L: 10%	3.07	0.33	0.11	x1.52	x1.39
	D: 50 ms	149.40	5.85	0.04	x1.20	x0.88
	D: 250 ms	31.16	0.80	0.03	x1.22	x1.02
	D: 350 ms	22.35	0.69	0.03	x1.19	x1.01
Netbird relay	Native	215.26	4.31	0.02	x3.57	x3.30
	L: 1%	42.02	0.66	0.02	x5.03	x4.46
	L: 5%	9.04	0.57	0.06	x4.35	x3.87
	L: 10%	2.49	0.20	0.08	x1.88	x1.71
	D: 50 ms	61.99	6.01	0.10	x2.90	x2.13
	D: 250 ms	24.38	2.17	0.09	x1.56	x1.31
	D: 350 ms	18.21	1.60	0.09	x1.46	x1.25
ZeroTier direct	Native	303.01	6.65	0.02	x2.54	x2.34
	L: 1%	89.29	1.40	0.02	x2.37	x2.10
	L: 5%	13.45	1.36	0.10	x2.92	x2.60
	L: 10%	3.12	0.34	0.11	x1.50	x1.36
	D: 50 ms	155.60	3.53	0.02	x1.15	x0.85
	D: 250 ms	31.89	0.62	0.02	x1.19	x1.00
	D: 350 ms	22.17	0.23	0.01	x1.20	x1.02

(Continues)

TABLE A1 (Continued)

Test	Cond	Mean	Std. Dev.	Coef. V.	Cmp. 1450B[a]	Cmp. 1230B[b]
ZeroTier relay	Native	256.94	5.51	0.02	x2.99	x2.76
	L: 1%	247.20	4.47	0.02	x0.85	x0.76
	L: 5%	85.59	6.08	0.07	x0.46	x0.41
	L: 10%	8.26	0.68	0.08	x0.57	x0.52
	D: 50 ms	150	22.62	0.15	x1.20	x0.88
	D: 250 ms	32.38	4.55	0.14	x1.17	x0.99
	D: 350 ms	23.63	3.86	0.16	x1.12	x0.96

Abbreviations: Coef. V., coefficient of variation; Cond, conditions; D, packet delay; L, packet loss; Std. Dev, standard deviation.
[a] Comparison to the baseline of MTU 1450B.
[b] Comparison to the baseline of MTU 1280B.

TABLE A2 UDP throughput results.

Test	Cond	Mean	Std. Dev.	Coef. V.	Cmp. 1450B[a]	Cmp. 1230B[b]
Headscale direct	Native	285.08	22.22	0.08	x2.66	x2.27
	L: 1%	283.90	23.72	0.08	x2.63	x2.25
	L: 5%	289.20	24.20	0.08	x2.49	x2.11
	L: 10%	284.30	22.81	0.08	x2.37	x2.00
	D: 50 ms	187.50	0.53	0.00	x1.19	x1.00
	D: 250 ms	37.12	0.26	0.01	x1.19	x1.00
	D: 350 ms	26.57	0.05	0.00	x1.19	x1.00
Headscale relay	Native	267.70	1.84	0.01	x2.83	x2.41
	L: 1%	267.00	2.05	0.01	x2.79	x2.40
	L: 5%	100.53	5.38	0.05	x7.15	x6.07
	L: 10%	8.57	1.48	0.17	x78.68	x66.37
	D: 50 ms	150.80	13.22	0.09	x1.48	x1.24
	D: 250 ms	35.81	2.24	0.06	x1.23	x1.04
	D: 350 ms	24.06	1.99	0.08	x1.31	x1.11
Netbird direct	Native	508.26	25.98	0.05	x1.49	x1.27
	L: 1%	491.30	27.86	0.06	x1.52	x1.30
	L: 5%	470.20	28.35	0.06	x1.53	x1.30
	L: 10%	457.50	14.58	0.03	x1.47	x1.24
	D: 50 ms	97.26	20.19	0.21	x2.29	x1.93
	D: 250 ms	12.67	3.43	0.27	x3.49	x2.94
	D: 350 ms	7.35	1.84	0.25	x4.29	x3.62
Netbird relay	Native	289.51	3.42	0.01	x2.62	x2.23
	L: 1%	286.70	2.58	0.01	x2.60	x2.23
	L: 5%	282.60	2.91	0.01	x2.54	x2.16
	L: 10%	275.80	2.04	0.01	x2.45	x2.06
	D: 50ms	183.70	1.06	0.01	x1.21	x1.02
	D: 250ms	36.58	0.58	0.02	x1.21	x1.02
	D: 350ms	26.05	0.77	0.03	x1.21	x1.02

(Continues)

TABLE A2 (Continued)

Test	Cond	Mean	Std. Dev.	Coef. V.	Cmp. 1450B[a]	Cmp. 1230B[b]
ZeroTier direct	Native	347.41	14.96	0.04	x2.18	x1.86
	L: 1%	347.80	10.55	0.03	x2.14	x1.84
	L: 5%	333.30	35.82	0.11	x2.16	x1.83
	L: 10%	337.56	15.13	0.04	x2.00	x1.69
	D: 50 ms	183.11	8.21	0.04	x1.22	x1.02
	D: 250 ms	36.43	0.57	0.02	x1.21	x1.02
	D: 350 ms	25.59	0.39	0.02	x1.23	x1.04
ZeroTier relay	Native	227.09	33.23	0.15	x3.34	x2.85
	L: 1%	231.00	6.27	0.03	x3.23	x2.77
	L: 5%	92.30	9.08	0.1	x7.79	x6.61
	L: 10%	8.02	1.37	0.17	x84.08	x70.93
	D: 50 ms	92.76	7.97	0.09	x2.40	x2.02
	D: 250 ms	39.27	3.05	0.08	x1.12	x0.95
	D: 350 ms	28.51	2.13	0.07	x1.11	x0.93

Abbreviations: Coef. V., coefficient of variation; Cond, conditions; D, packet delay; L, packet loss; Std. Dev, standard deviation.

[a] Comparison to the baseline of MTU 1450B.

[b] Comparison to the baseline of MTU 1280B.