Scalability evaluation of VPN technologies for secure container networking

1st Tom Goethals
Department of Information Technology
Ghent University - imec, IDLab
Gent, Belgium
togoetha.goethals@UGent.be

2nd Dwight Kerkhove
Department of Information Technology
Ghent University - imec, IDLab
Gent, Belgium
dwight.kerkhove@UGent.be

3rd Bruno Volckaert
Department of Information Technology
Ghent University - imec, IDLab
Gent, Belgium
bruno.volckaert@UGent.be

4th Filip De Turck
Department of Information Technology
Ghent University - imec, IDLab
Gent, Belgium
filip.deturck@UGent.be

***Abstract*—For years, containers have been a popular choice for lightweight virtualization in the cloud. With the rise of more powerful and flexible edge devices, container deployment strategies have arisen that leverage the computational power of edge devices for optimal workload distribution. This move from a secure data center network to heterogenous public and private networks presents some issues in terms of security and network topology that can be partially solved by using a Virtual Private Network (VPN) to connect edge nodes to the cloud. In this paper, the scalability of VPN software is evaluated to determine if and how it can be used in large-scale clusters containing edge nodes. Benchmarks are performed to determine the maximum number of VPN-connected nodes and the influence of network degradation on VPN performance, primarily using traffic typical for edge devices generating IoT data. Some high level conclusions are drawn from the results, indicating that WireGuard is an excellent choice of VPN software to connect edge nodes in a cluster. Analysis of the results also shows the strengths and weaknesses of other VPN software.**

***Index Terms*—Containers, Microservices, Virtual private network, VPN, Edge network, Network security**

I. Introduction

In recent years, containers have quickly gained popularity as a lightweight virtualization alternative to virtual machines, their main advantages being limited resource requirements and fast spin-up times [2]. Lately, edge devices used in IoT applications have become powerful enough to smoothly run containers and microservices, while remaining small and flexible enough to be used almost anywhere. This has triggered a trend of deploying containerized applications to edge devices, taking advantage of both centralized control and geographically widespread devices for a more efficient distribution of workloads [4]. To aid with the deployment and operation of containers on edge devices, a Virtual Private Network (VPN) can be useful, for the following reasons:

- Securing communication between nodes becomes more important when leaving the confines of the cloud. While the connections between orchestrator nodes and service endpoints are often secured in various ways by default, it can not be assumed that this is always the case. A VPN can provide a base layer of security for all communications, ideally with little to no performance overhead.
- Unlike cloud infrastructure, networks containing edge devices are not usually well organized and homogeneous. This means the network could be hidden behind a router, node IP addresses are not predictable, existing port mappings could interfere with container requirements, etc. While technologies such as UPnP [3] can solve some of these problems, they can also introduce new security problems. It would be better to avoid the problems with edge networks altogether by using a VPN. In the cloud, only the VPN server needs to be publicly available. This makes it possible to hide critical services from public scrutiny, while still allowing edge devices in the VPN to access them over a secure connection. In short, a VPN can ensure a homogeneous networking environment in which both the container orchestrator and deployed containers can allocate required ports, assign IP addresses and modify routing tables without interference from other devices or programs.

Modern clusters consist of thousands of nodes on which containers can be deployed [1]. Therefore, in order to build a cluster using a VPN, any suitable VPN software must be able to handle a large number of simultaneous connections and packets with minimal performance degradation. The goal of this paper is to evaluate recent VPN software for its ability to scale with the size of the cluster using it as an overlay network, while also examining the effects of network degradation on communication between nodes in a cluster connected through a VPN. While some of the conclusions are valid for VPN networks in general, the tests are primarily aimed at edge networks, with a lot of devices generating IoT traffic consisting of very small packets.

Section II presents existing research related to the topic of this paper. Section III lists the different VPN software that is benchmarked, while section IV details the test setup and test scenarios used to gauge scalability. The results are presented and discussed in section V, with suggestions for future work in section VI. Lastly, section VII summarizes the conclusions and lists suggestions on which VPN to use.

II. Related Work

Many studies exist on the concept of shifting workloads between the cloud and edge hardware. This trend began with edge offloading [7] and cloud offloading [5], evolving into osmotic computing [4]. Some studies focus particularly on security for osmotic computing and end-to-end security between the edge and the cloud [11, 13].

VPNs are an old and widely used technology. Recent state of the art studies appear to be non-existent, but older ones are still informative [17]. Some studies deal with the security aspects of a VPN [20], while many others focus on the throughput performance of VPNs [16, 14]. However, no studies seem to compare OpenVPN [23] to newer VPNs such as WireGuard [21].

While studies exist on using overlay networks in osmotic computing [8], they deal mostly with container network overlays such as Flannel and Weave [12] which are integrated into Kubernetes. Others present a custom framework, for example Hybrid Fog and Cloud Interconnection Framework [18], which also gives a good overview of the challenges of connecting edge and cloud networks. To the best of our knowledge, no work exists on explicitly using a VPN as an overlay network to connect edge and cloud networks, nor is there existing work on testing the scalability of a VPN to do so.

III. VPN Software

In this section, an overview is given of all the VPN software chosen for this paper. For each VPN solution, a Docker container image was created so it could be launched from a Kubernetes pod. As much default configuration was used as possible. The code and configuration for the Docker container images is made available on Github[1].

A. OpenVPN

OpenVPN [23] is a widely used VPN solution, first released in May 2001. It is still under active development, with version 2.4.7 being released as of February 2019. For the tests in this study, version 2.4.6 was used. It uses open encryption standards and offers a wide range of options, while being able to use both UDP and TCP for its transport layer. A weak point of OpenVPN is that it is single threaded and thus entirely dependent on the speed of a single processor core, no matter how powerful a system is. For the test, the standard configuration is used. Traffic is encapsulated in UDP packets over a TUN device.

B. WireGuard

WireGuard [21] is a relatively new VPN solution. It does not have an official stable release yet, and is still under heavy development. Its current version is v0.0.20190406, but the version used for the tests is 0.0.20180613. As of March 2019, it has been sent out for review several times, aiming to be mainlined into the Linux 5.2 kernel [15]. A main feature of WireGuard is that it is designed to be non-chatty, only sending messages when required. Note that in this paper WireGuard-go [19] is used, which is a Golang implementation of WireGuard. Performance may differ from the main WireGuard version, especially if it is accepted into the Linux kernel.

For the test, the standard configuration is used. Some extra work is required to set up interfaces and routing rules, which is done by default in other software such as OpenVPN and ZeroTier.

C. ZeroTier

ZeroTier [22] is an established VPN solution developed by ZeroTier, Inc. First developed in 2011, it is currently at version 1.2.12, which is also used for the tests. ZeroTier is still under active development, and has both paid and free solutions. The "Zero" part of its name comes from the fact that it requires zero configuration by default. This is achieved by having a number of root servers, called the "Earth" and managed by ZeroTier, that fulfill a similar role as DNS servers to the ZeroTier network. However, users can also define their own root servers using "Moons" in order to decrease dependency on the ZeroTier cloud infrastructure and improve performance. Its design also gives ZeroTier the advantage that no endpoints need to be publicly available, as long as they are still accessible through some public IP address via NAT.

For the test, a ZeroTier network controller was set up using a script to simplify the creation of VPN clients through scripts. While there is a minimal reliance on ZeroTier cloud infrastructure, the Earth root servers are still used to look up the address of the test network controller by its ZeroTier name.

D. Tinc

Tinc [24] is a VPN solution that predates even OpenVPN, with an initial release in November 1998. The current version, 1.0.35, was released in October 2018, so it is still under active development. Version 1.0.35 is also the one used for the tests. Like OpenVPN, it relies heavily on open standards. However, unlike OpenVPN, it has full mesh routing by default, which can make it more efficient in large networks with large amounts of client-to-client traffic.

For the test, standard configuration was used. Like OpenVPN, transport uses UDP via a TUN device.

E. SoftEther VPN

SoftEther VPN [25] is a relatively new VPN solution. Its first release dates from January 2014, with the latest release being version 4.29 as of February 2019.

SoftEther is a multi-protocol VPN, with modules for OpenVPN, L2TP/IPSec, MS-SSTP and its own SoftEther VPN

[1] https://github.com/drake7707/secure-container-network

protocol. By default it uses the SoftEther VPN protocol, which emulates Ethernet over HTTPS. The advantage of using HTTPS to tunnel VPN traffic is that it makes it easier to bypass firewalls, since HTTPS ports are often freely accessible.

Other than being multi-protocol, SoftEther has a wide range of features, including a high availability setup for its server endpoints.

IV. BENCHMARKING SETUP

The tests are performed on the IDLab Virtual Wall installation, reserving machines with identical hardware and in the same geographical location for each test. Each machine has two Quad core Intel E5520 processors at 2.2GHz and 12GiB RAM.

A total of 8 machines are provisioned with Ubuntu 16.04 and Docker 18.06. Using these machines, a Kubernetes v1.11 cluster is created with 1 master node and 7 worker nodes as a basis for each test. Kubernetes is used to easily distribute a large number of VPN client containers over the entire cluster to simulate a VPN network with many independent clients. The Kubernetes version should not have any impact on the results up to and including v1.14. Observations during testing showed that deploying over 120 VPN client containers on a single node results in errors because the container processes fail to allocate all required resources. However, when using Kubernetes this is not a problem, since it has a built-in limit of 110 pods per node by default. Therefore, the VPN containers deployed by Kubernetes for the tests will not run into the observed problems with overdeployment.

All tests follow the same basic premise, using pods that contain a single container with a VPN client and a configurable script. A number of pods are deployed to the nodes in the cluster by creating a new deployment in Kubernetes, as shown in Fig. 1. The actual number of deployed pods varies depending on the test. Once a VPN connection is established, the script in the pod starts calling a REST service located on the master node using curl, every 250ms for 15 minutes. The REST request has no body, it is an empty GET request. The response is a simple “OK” with a 200 HTTP code. Using these short and static request/response messages ensures that the connectivity of all VPNs is tested rather than network throughput. This approach also has the advantage that it closely emulates IoT sensor traffic usually present in edge networks, where short bursts of sensor data are pushed to a central broker. The pods do not perform any processing of the results, rather they simply log the start timestamp, end timestamp and curl exit code of each REST call to a pod-named output file on a host-mounted network share to be processed later.

Because the goal of the tests is to measure VPN scalability with no other factors involved, the VPN server and REST service containers on the master node are started outside Kubernetes. This ensures that none of the VPN traffic goes through the Kubernetes pod network, but rather that it travels directly between VPN endpoints.

To ensure only valid data was processed, the data was filtered both at the start and the end of each test:

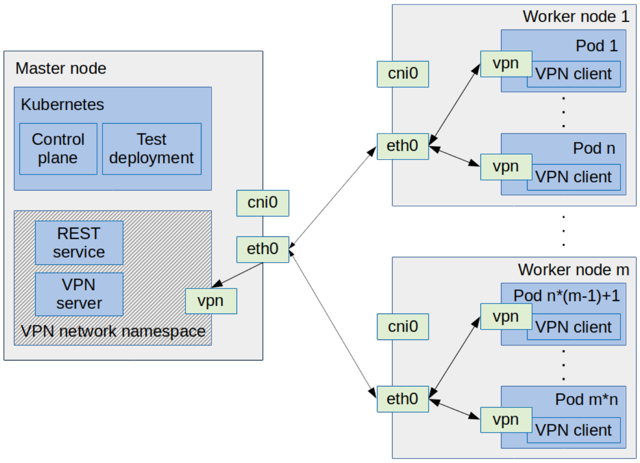

Fig. 1. Overview of the test setup with m nodes and n pods per node. All of the traffic goes via the VPN, none via the pod network (cni0).

- To remove data generated before all pods were running, only data recorded after the latest first timestamp of all pod output files is used.
- To remove data generated after the test is shut down, the last string of failures at the end of each pod output file is removed. In theory, this could remove some valid failures, but the total number of calls for each pod is large enough that it should not make any meaningful difference.

A. Scalability test

The scalability test aims to determine how well VPN software can handle large amounts of connections and service calls. To achieve this, a number of pods are deployed via Kubernetes that perform REST calls as described above. The number of deployed pods varies from 50 to 750, in steps of 100. The results from the test are aggregated and analyzed to determine how the response times and failure rates (error codes) vary with an increasing amount of pods.

B. Network degradation test

In the real world there is always some amount of packet loss and latency, especially when dealing with the edge and IoT devices. Therefore, it is important that any VPN can remain functional despite significant network degradation.

To determine the resilience of VPN software against network degradation, both packet loss and latency are examined independently. Both conditions are imposed through NetEM on the host network interface. Because the degradation affects all traffic including VPN control messages, this method gives a good indication of how well a VPN can keep working under any set of network conditions.

This test is performed with 100 pods distributed evenly over the 7 worker nodes. Latency is varied from 0 to 5000ms in 1000ms jumps. Out of practical considerations, any requests taking longer than 10 seconds are terminated and considered a failure. This influences the failure rate to some degree, but since the same rule is applied across all tested VPNs, the comparison is still valid. Packet loss is varied from 0% to 100%. While this range may seem extreme, it is interesting to see how VPNs react to the full spectrum of packet loss.

V. Results

In this section the results from the tests are presented. To keep this section organized, the results for the scalability tests have been divided into subsections for response time and failure rate. Similarly, the network degradation tests have been split into subsections for latency and packet loss.

Note that for all charts with whiskers, the whiskers indicate the 25th and 75th percentiles of the results, while the data points represent the median cases. The 75th percentile is used for practical reasons; higher percentiles produce extremely large whisker ranges, making it harder to interpret the charts. The 25th percentile is chosen for symmetry; the difference between the 1st and 25th percentile is far less significant than between the 75th and 100th.

A. Scalability - response time

Fig. 2 shows the response times of the scalability test on a logarithmic scale. For the same reason, both charts have been truncated at 650 pods instead of 750. A striking observation is that the response times of both Tinc and SoftEther increase by an order of magnitude between 250 and 350 clients, making them over 50 times higher than those of other VPNs. At less than 350 clients, Tinc is already being outpaced by OpenVPN, but is still in the same order of magnitude. SoftEther starts out with response times twice as high as those of OpenVPN, and ends up with response times almost 150 times higher than those of OpenVPN at 650 clients. It is likely that some common feature or design decision of Tinc and SoftEther is causing this performance problem.

OpenVPN, WireGuard and ZeroTier are evenly matched for 450 VPN clients or less. Even the 25th to 75th percentile ranges are similar, though the highest ZeroTier response times start to go up quickly around 250 clients. For more than 450 clients, a few trends become clear that are useful for large scale edge networks. First, increasing VPN traffic does not seem to affect WireGuard response times at all. Even the results at 650 clients are still in line with those at 50 clients. OpenVPN response times shift from a slow, linear increase to a quadratic curve. ZeroTier is on a solid quadratic curve from the start, while the slowest responses quickly escape the chart altogether. In short, WireGuard is the clear winner of this test, while OpenVPN is a close second. However, if the test were to continue to thousands of clients, WireGuard may become the only useful choice.

B. Scalability - failure rate

Fig. 3 shows the failure rates of the scalability test. The results are strikingly similar to the REST response times.

Tinc failure rates are an order of magnitude above those of other VPNs, but the curve is more pronounced than in the response times chart. Even with 50 clients, Tinc failure rate is 10 times higher than that of OpenVPN and ZeroTier. It is possible that Tinc is more suited to high throughput applications in smaller networks, but that topic is outside the scope of this paper. SoftEther, while not having excellent results, stays close to OpenVPN performance until its failure rates start to increase quickly around 450 clients.

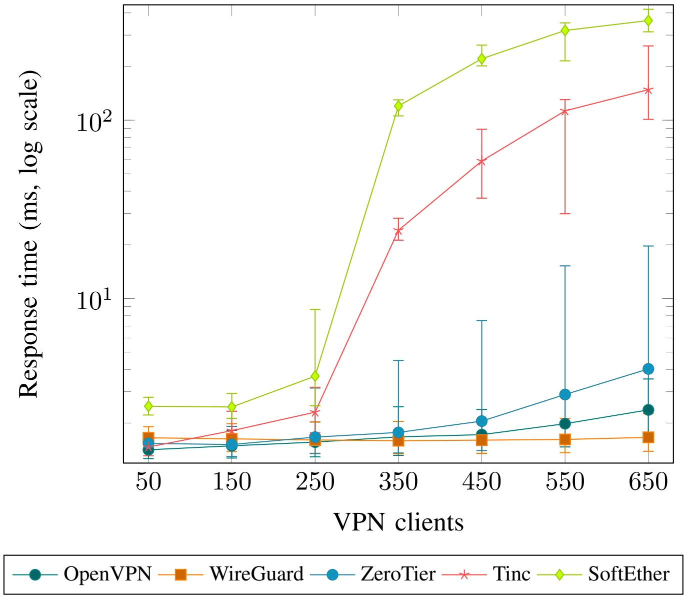

Fig. 2. Evolution of response time for an increasing number of VPN clients, represented on a logarithmic scale.

As the rest of the VPNs go, WireGuard has the best results, showing little to no increase in failed requests as VPN traffic and the number of clients increases. OpenVPN appears to follow a linear trend, with ZeroTier following closely until its performance once again degrades according to a quadratic curve around 450 deployed clients.

C. Network degradation - packet loss

Fig. 4 shows the response time for REST requests under each VPN, for increasing amounts of packet loss. All the results in this chart are very closely matched, with even the 25th to 75th percentile ranges lining up in most cases. The results only start to differentiate for 70% packet loss or more. WireGuard performance degrades heavily when going from 90% to 100% packet loss, seemingly making it the slowest VPN in this test. Tinc appears to win this test, with the lowest median time and a relatively small range for its results.

Considering the invariance of the results at normal amounts of packet loss, 0% to 20%, it stands to reason that increasing the number of clients within this range will yield the exact same results as Fig. **??** through Fig. 3 have shown. However, more tests are required to properly confirm this.

OpenVPN and SoftEther results cannot be shown beyond the 80% packet loss category, the reason for which becomes clear when looking at Fig. 5. This chart shows the failure rates for the same packet loss test, and also suggests an alternate interpretation for some of the apparent effects in Fig. 4. While

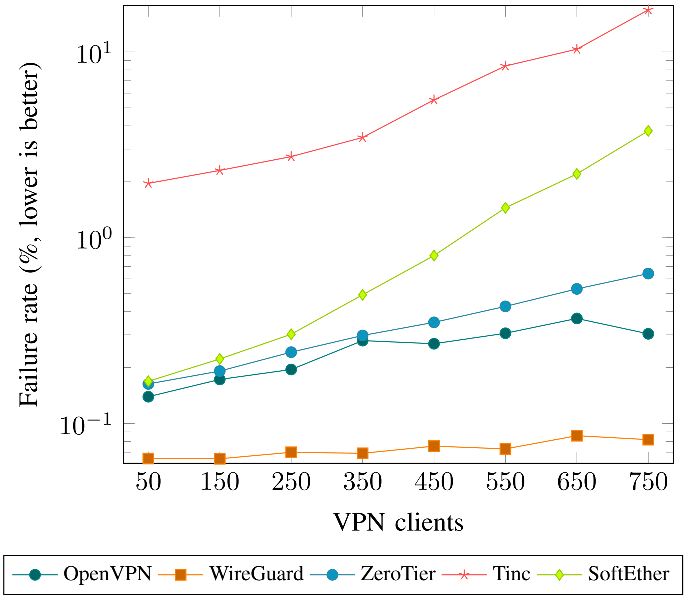

Fig. 3. Evolution of failure rate for an increasing number of VPN clients.

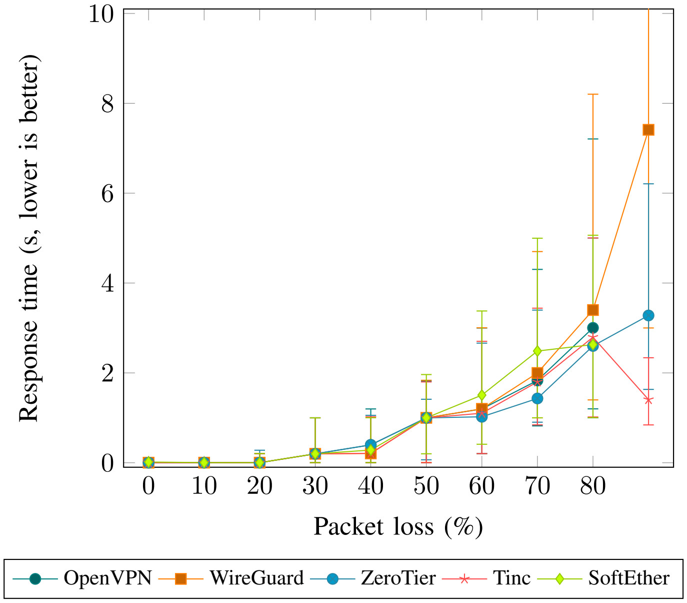

Fig. 4. Response time of REST calls from 100 VPN clients for increasing packet loss

ZeroTier and WireGuard manage to keep pushing at least some traffic through until packet loss hits 100%, Tinc, SoftEther and OpenVPN failure rates increase much faster. Starting around 90% packet loss, OpenVPN and SoftEther both have a 100% failure rate. Tinc on the other hand, still has a 0.004% success rate, but the extremely low sample size makes the response time shown in Fig. 4 very unreliable. WireGuard and ZeroTier still have a success rate of about 10% to 20%, so that data is reliable.

While there is a lot of nuance in the results for 70% to 100% packet loss, the results for the lower end of 0% to 20%

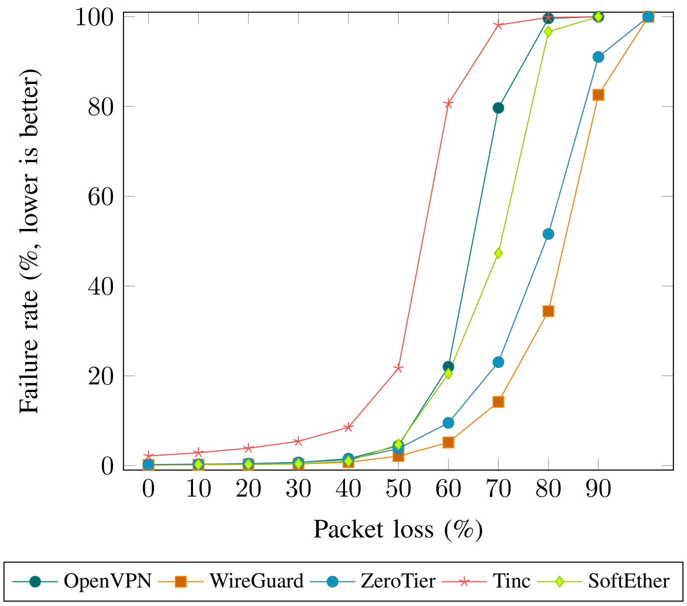

Fig. 5. Failure rate of REST calls from 100 VPN clients for increasing packet loss

are easier to interpret. There is little difference between the different VPNs in this range, save for Tinc, which is known to have a higher failure rate from the results of the previous tests. There is no clear winner in this test; both WireGuard and ZeroTier are good options, with WireGuard having slightly higher response times but lower failure rates.

D. *Network degradation - latency*

Fig. 6 shows the failure rate for each VPN for increasing amounts of network latency.

WireGuard has the best failure rate, showing only a slow increase in failure rates as latency goes up. Even at 5000ms latency, still only around 3% of the requests fail. Note that due to the way the test was set up, WireGuard failure rate is 100% as soon as latency goes over 5000ms. ZeroTier and SoftEther are very close in performance. Their failure rates start out almost equal to those of WireGuard, but they increase faster as latency goes up. SoftEther failure rates shoot up to almost 50% at 4000ms, while ZeroTier manages to stay around 18%. Tinc, which by now has firmly established a somewhat high failure rate, degrades quickly once latency is higher than 1000ms, with a failure rate of 83% at 2000ms latency.

OpenVPN holds the middle ground between ZeroTier and Tinc, with error rates over twice as high as those of ZeroTier. OpenVPN shows slightly better performance than SoftEther at 4000ms latency, but at that point almost half of all requests fail under both OpenVPN and SoftEther.

As with the packet loss test, WireGuard comes out on top with very low failure rates, while ZeroTier is a close second.

VI. FUTURE WORK

While the results in this paper provide a glimpse into the scalability of VPNs to connect edge nodes to a cluster, many

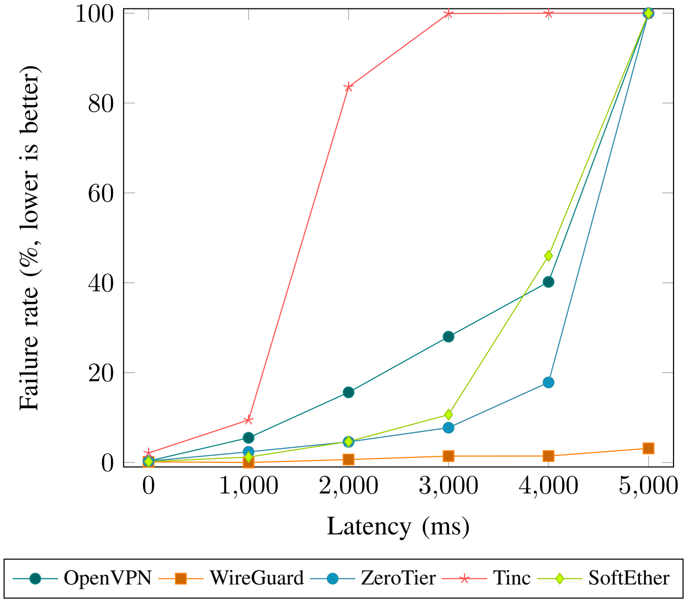

Fig. 6. Failure rate of REST calls from 100 VPN clients for increasing latency

more topics are open for research.

For starters, much more VPN software exists than has been benchmarked. Adventurous researchers could attempt to map the scalability of all known VPN software, though this work will be hampered by a quickly changing landscape of possible candidates and versions.

While the results of this study provide insight on the high-level behavior of VPN software, a lot of the parameter space was deliberately not examined. Evaluating the full parameter space would be prohibitively time-consuming. However, there may still be interesting behavior to discover. For example, there is no way to predict how the results from the packet loss and latency tests would change with a varying number of VPN clients.

Furthermore, no attempt has been made to optimize the configuration of each VPN to improve performance. A technical study which analyses VPN designs and configuration options could yield important insight into making a VPN that is ideal for use with mixed cloud-edge clusters.

It is possible that network topology and the number of VPN clients per node has a significant influence on the scalability of a VPN network. To confirm this, the tests would have to be repeated using hundreds of physical machines.

Another interesting topic is to see how a VPN network could be extended by using multiple VPN servers in a cluster. For example, SoftEther has a clustering function [26] that could be combined with a Kubernetes high availability setup to provide a suitable VPN network topology for a fault tolerant cluster.

Lastly, the results showed that up to 750 VPN clients is not a problem for WireGuard. Studies focusing on one specific VPN solution could be useful to determine its scalability limits.

VII. Conclusion

The goal of this paper is to provide insight on the scalability of using VPNs to connect nodes in edge and IoT clusters, and to examine the effects of network degradation on VPNs used for this purpose. To that end, a number of benchmarks are described. OpenVPN, WireGuard, ZeroTier, Tinc and SoftEther are set up with a default configuration and subjected to the described benchmarks.

The results for WireGuard are the best across all tests. In some cases the results are remarkable, such as an almost unchanging response time and failure rate for REST requests with an increasing number of clients, where other VPN solutions tend to have quadratic scaling. In other cases, the difference is less pronounced. In the packet loss tests for example, the response times for requests over WireGuard are mostly on par with other VPN setups, but the failure rates are about 10% to 30% lower than those for ZeroTier, which holds the second place in that scenario.

Tinc seems to be the least suited to the kind of setup used for the tests, having the slowest response time and highest failure rate with a large amount of clients, while also being most susceptible to network degradation.

The other solutions have varying results in all of the tests. In the scalability tests, OpenVPN holds second place, while ZeroTier comes in third. SoftEther is usually near Tinc in terms of performance. Looking at the network degradation tests, SoftEther and OpenVPN switch places while ZeroTier remains in second place.

Some topics for future work are discussed that could lead to further insight into the scalability and performance of VPNs. Further research into these topics could assist development of a VPN optimized for use with edge and IoT clusters.

VIII. Acknowledgment

The research in this paper has been funded by Vlaio by means of the FLEXNET research project.

References

[1] Kubernetes - Building large clusters, https://kubernetes.io/docs/setup/cluster-large/

[2] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy and Y. C. Tay, Containers and Virtual Machines at Scale: A Comparative Study, Middleware ’16 Proceedings of the 17th International Middleware Conference, DOI: 10.1145/2988336.2988337

[3] Abul Ahsan and Mahmudul Haque, UPnP Networking: Architecture and Security Issues, https://www.researchgate.net/publication/242305803_UPnP_Networking_Architecture_and_Security_Issues

[4] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana and Rajiv Ranjan, Osmotic Computing: A New Paradigm for Edge/Cloud Integration, IEEE Cloud Computing (Volume: 3, Issue: 6, Nov.-Dec. 2016), DOI: 10.1109/MCC.2016.124

[5] Karthik Kumar and Yung-Hsiang Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, Computer April 2010, pp. 51-56, vol. 43, DOI: 10.1109/MC.2010.98

[6] Ahsan Morshed, Prem Prakash Jayaraman, Timos Sellis, Dimitrios Georgakopoulos, Massimo Villari and Rajiv Ranjan, Deep Osmosis: Holistic Distributed Deep Learning in Osmotic Computing, IEEE Cloud Computing (Volume: 4, Issue: 6, November/December 2017), DOI: 10.1109/MCC.2018.1081070

[7] Pavel Mach and Zdenek Becvar, Mobile Edge Computing: A Survey on Architecture and Computation Offloading, IEEE Communications Surveys & Tutorials (Volume: 19 ,Issue: 3 ,third quarter 2017), DOI: 10.1109/COMST.2017.2682318

[8] Alina Buzachis, Antonino Galletta, Lorenzo Carnevale, Antonio Celesti, Maria Fazio and Massimo Villari, Towards Osmotic Computing: Analyzing Overlay Network Solutions to Optimize the Deployment of Container-Based Microservices in Fog, Edge and IoT Environments, 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), DOI: 10.1109/CFEC.2018.8358729

[9] Jose Santos, Tim Wauters, Bruno Volckaert and Filip De Turck, Resource Provisioning in Fog Computing: From Theory to Practice, Sensors 19(10):2238 DOI: 10.3390/s19102238

[10] Daniele Santoro, Daniel Zozin, Daniele Pizzolli, Francesco De Pellegrini and Silvio Cretti, Foggy: a platform for workload orchestration in a Fog Computing environment, 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), DOI: 10.1109/CloudCom.2017.62

[11] Deepak Puthal, Surya Nepal, Rajiv Ranjan and Jinjun Chen, Threats to Networking Cloud and Edge Datacenters in the Internet of Things, IEEE Cloud Computing (Volume: 3, Issue: 3, May-June 2016), DOI: 10.1109/MCC.2016.63

[12] Cluster networking, https://kubernetes.io/docs/concepts/cluster-administration/networking/

[13] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, Lydia Chen and Rajiv Ranjan, Software Defined Membrane: Policy-Driven Edge and Internet of Things Security, IEEE Cloud Computing (Volume: 4, Issue: 4, July/August 2017), DOI: 10.1109/MCC.2017.3791014

[14] I. Kotuliak, P. Rybr and P. Trchly, Performance comparison of IPsec and TLS based VPN technologies, 2011 9th International Conference on Emerging eLearning Technologies and Applications (ICETA), DOI: 10.1109/ICETA.2011.6112567

[15] WireGuard Sent Out Again For Review, Might Make It Into Linux 5.2 Kernel, https://www.phoronix.com/scan.php?page=news_item&px=WireGuard-V9-Maybe-Linux-5.2

[16] Frederic Pohl and Hans Dieter Schotten, Secure and Scalable Remote Access Tunnels for the IIoT: An Assessment of openVPN and IPsec Performance, ESOCC 2017: Service-Oriented and Cloud Computing pp 83-90

[17] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba, Network virtualization: state of the art and research challenges, IEEE Communications Magazine (Volume: 47, Issue: 7, July 2009), DOI: 10.1109/MCOM.2009.5183468

[18] Rafael Moreno-Vozmediano, Ruben S. Montero, Eduardo Huedo and Ignacio M. Llorente, Cross-Site Virtual Network in Cloud and Fog Computing, IEEE Cloud Computing (Volume: 4, Issue: 2, March-April 2017), DOI: 10.1109/MCC.2017.28

[19] About WireGuard-go, https://git.zx2c4.com/wireguard-go/about/

[20] H. Hamed, E. Al-Shaer and W. Marrero, Modeling and verification of IPSec and VPN security policies, 13TH IEEE International Conference on Network Protocols (ICNP'05), DOI: 10.1109/ICNP.2005.25

[21] Jason A. Donenfeld, WireGuard: Next Generation Kernel Network Tunnel, https://www.wireguard.com/papers/wireguard.pdf

[22] ZeroTier manual, https://www.zerotier.com/manual.shtml

[23] Jason A. Donenfeld, Securing Remote Access Using VPN, https://openvpn.net/whitepaper/

[24] Ivo Timmermans and Guus Sliepen, Setting up a Virtual Private Network with tinc, https://www.tinc-vpn.org/documentation/tinc.pdf

[25] Daiyuu Nobori, Design and Implementation of SoftEther VPN, https://www.softether.org/@api/deki/files/399/=SoftEtherVPN.pdf

[26] SoftEther clustering, https://www.softether.org/4-docs/1-manual/3._SoftEther_VPN_Server_Manual/3.9_Clustering