Network Federation for Inter-cloud Operations

Johannes Köstler, Sven Gebauer, Hans P. Reiser

Network Federation for Inter-Cloud Operations

Johannes Köstler[1][0000−0001−9579−9908], Sven Gebauer[2][0000−0002−6092−7305], and Hans P. Reiser[1][0000−0002−2815−5747]

[1] University of Passau, Germany {jk,hr}@sec.uni-passau.de
[2] gebauers@fim.uni-passau.de

Abstract. This paper introduces the NetFed network federation approach, which enables inter-cloud operations on the basis of shared overlay networks. This allows to match the particular application needs to the most suitable infrastructure provider. The agent-based federation approach utilizes WireGuard and GRE to deliver a flexible and transparent layer 2 or layer 3 overlay network, while maintaining data integrity and confidentiality. The evaluation shows that our prototype can be deployed to arbitrary cloud platforms and benefits from a low traffic and processing overhead.

Keywords: Network federation · Network virtualization · Virtual private network · Cloud orchestration · Inter-cloud operation

1 Introduction

Cloud federation is the practice of interconnecting two or more private or public cloud infrastructures. This practice allows clients to optimize the use of cloud infrastructures by using resources of the best cloud service provider in terms of costs, flexibility, availability, legal constraints, and particular technological needs. By using multiple different cloud platforms, a customer can avoid potential vendor lock-in, and the combination of public cloud and private in-house infrastructure can offer advantages in terms of security and privacy.

One of the major challenges with cloud federation is the federation of networks: The transparent distribution across multiple cloud platforms, in particular in the IaaS service model, requires a distributed virtual network that behaves much like a virtual network within a single cloud platform. Most cloud platforms offer some ways of connecting external machines to their networks through virtual private networks (VPNs), but protocols and configurations differ significantly between various cloud platforms. These differences make a combination of several cloud networks non-trivial. Especially under the aspects of flexibility and vendor lock-in, the question arises whether and how it is possible for a cloud user to federate multiple cloud networks in a generic way that allows for addition and removal of network segments.

In this paper, we explore existing approaches to cloud federation and present a novel solution based on WireGuard tunneling and distributed agents for automated configuration. Our research focuses on a number of goals: (i) Optimizing

performance, i.e., minimizing latency, maximizing throughput, and avoiding potential central bottlenecks in the over-all architecture; (ii) federation at the data link layer, in order to enable the use of broadcast, unknown-unicast, and multicast traffic (BUM traffic) as well as the use of IPv4 and IPv6 traffic; (iii) easy-to-use and transparent deployment on any private and public cloud infrastructure; (iv) security, in particular prevention of unauthorized access from outside, as well as confidentiality and integrity for any network traffic forwarded between cloud endpoints; (v) support for efficient multiple, isolated virtual networks via a single inter-cloud tunnel.

The remainder of this paper is structured as follows. The next section discusses existing approaches and related work in the field of network federation. In Section 3 we systematically explore and compare existing technology for network tunnel protocols that potentially can be used for network federation, and justify why WireGuard is the best choice for our purposes. Section 4 presents our NetFed architecture that enables automated federation of isolated network segments based on federation agents. Section 5 presents evaluation results, before the paper closes with a short conclusion.

2 Related Work

Federating isolated networks for resource aggregation on the basis of VPNs is a common concept. Wood et al. [24] define the hybrid cloud use case as virtual private cloud (VPC), in which VPN technology connects computing resources from cloud sites to the enterprise site. The cloud provider deploys custom edge routers that can be configured by a network manager to serve as VPN tunnel endpoint in the cloud network. The authors also propose a reference architecture called CloudNet that uses multiprotocol label switching (MPLS) and virtual local area networks (VLANs) to provide isolated tunnels for each cloud customer. With the emergence of software defined networks (SDNs) [12], the provision and control of such cloud gateways was centralized and simplified with OpenFlow [20]. Nowadays, all major cloud providers offer some sort of VPC service for site-to-site and client-based point-to-site connections. These VPCs can provide benefits with respect to scalability and interoperability for confined use cases [19], but are restricted to a single cloud provider and do not necessarily provide the full networking functionality. For instance, Amazon Web Services (AWS) only transports traffic based on a proprietary routing mechanism and all address ranges used in a virtual network must be known to the AWS platform, what prevents BUM packets from being forwarded [21]. Google Cloud Platform (GCP), on the other hand, does currently only support IPv4 unicast traffic [5].

Vendor-specific solutions ultimately bear the risk of vendor lock-in. Tai et al. [23] propose cloud federation as a way to avoid this risk and underline the advantages arising from multi-cloud deployments. Many cloud federation architectures leverage SDNs to combine isolated clouds in virtual overlay networks [3, 10, 13, 16, 17]. Mechtri et al. [13] deploy Cloud Networking Gateways (CNG) as OpenFlow switches in each cloud, which are controlled by a central

CNG manager. Gateways provide different tunnel protocols like IPsec, OpenVPN, Generic Routing Extension (GRE) or OpenFlow as drivers. Gaul et al. [3] deploy the OpenVirteX (OVX) [1] network virtualization platform over multiple clouds. Each cloud provider runs the network hypervisor on its gateways, so that the SDN controllers of the tenants can provision multiple virtual SDNs over all participating clouds. OVX maps multiple virtual resources to physical resources and uses MAC and IP rewriting in the flows communicated to the underlying networking infrastructure. Levin et al. [10] also provide multi-cloud interoperability through inter-cloud networking based on federation agents. These agents are in fact OpenDaylight [14] controllers that run OpenDOVE (Distributed Overlay Virtual Network) to provide overlay networks and communicate with each other to established Virtual Extensible LAN (VXLAN) tunnels across multiple clouds. The agents are controlled by a central federation manager, which manages the cross-cloud networks of the cloud tenants. The BEACON framework proposes a similar architecture with an envisioned extension of OpenDOVE [16].

Another group of more use-case-driven approaches renounce the complexity of SDNs [4,9,11,15,25]. WAVNet [25] builds a virtual cloud by directly connecting common desktop computers on a layer 2 overlay. The peers are connected by tap devices tunneled over the Internet. Rendezvous servers are used to make peers known to each other and UDP hole punching accesses computers behind network address translation (NAT) services. Moreno-Vozmediano et al. [15] present a framework that connects devices of fog and cloud environments via tunnel agents controlled by a central management instance. The agents establish layer 2 or layer 3 tunnels using GRE and IPsec. Compared to connections without overlay, their prototype achieves 50% of the original throughput with a secure layer 2 overlay and almost 70% with a secure layer 3 overlay. The approach proposed by Kimmerlin et al. [9] has a similar structure, but instead of a central manager the overlay can be shaped by all participating agents. The agents provide layer 2 tunnels using VXLAN, GRE, or Geneve over IPsec. The resulting throughput stagnates at around 60-70% of the reference throughput depending on the used IPsec ciphers. Mansouri et al. [11] propose a network federation based on the WireGuard VPN and Terraform in a database evaluation use case. WireGuard agent VMs are provisioned using Terraform and establish layer 3 tunnels between arbitrary clouds. The solution does not support layer 2 networking and unfortunately there are no evaluations on the sole tunnel performance. Goethals et al. [4] provide cross-domain federation in orchestration environments by deploying OpenVPN containers in different networks. The deployment is integrated into the Docker orchestrator. However, the use of OpenVPN limits their throughput to barely 10% of the available bandwidth.

Summing up, it can be concluded that there exist very different solutions with distinctive characteristics. SDN-based solutions offer the greatest flexibility and can provide far more extensive functionality. However, they require heavyweighted controller and their large protocol stack adds additional complexity. There are various agent-based solutions that impress with their simplicity, but

not all support all networking functionality and their performances differ to a large extent.

3 Tunnel Protocols

For selecting the optimal tunnel protocol for our overlay networks, we compare a variety of modern and freely available VPN protocols. Proprietary protocols like Microsoft's Secure Socket Tunneling Protocol (SSTP) or rather dated protocols like Point-to-Point Tunneling Protocol (PPTP) are not considered. Besides the possibility to transport IP packets and Ethernet frames over the Internet, our main protocol requirements are low connection overhead and high throughput as well as encryption and authentication support for security purposes. We set up VPN tunnels with all these technologies to measure their performance, assess their security features, and gain insights about their usage. The results of this evaluation are shown in Table 1 and discussed in the following.

3.1 Forwarding

The private networks of the VPN protocols are provided at different network layers. OpenVPN and tinc support both the creation of virtual tap or tun devices in order to act as virtual Ethernet switches or virtual routers. SoftEther provides the same functionality, but follows its own terminology (virtual hub/L3 switch). ZeroTier offers emulated Ethernet networks only, whereas WireGuard and IPsec operate only on layer 3. In that case, an additional transport protocol is required to run Ethernet over the IP tunnel. Traditionally IPsec is often combined with L2TP, but we found GRE to be more light-weight and fully stateless. L2TP adds additional overhead through its session concept. OpenVPN and SoftEther follow a star topology with a central server, whereas the other support a full mesh topology. Link deactivation examines whether the internal routing can be controlled to favor certain links. This is only possible with IPsec and WireGuard, as the routing has to be implemented manually. A key requirement for the utilized VPN protocol is the ability to forward BUM traffic. IPsec and WireGuard on layer 3 have no built-in broadcast support. Thus, such traffic requires a layer 2 tunnel for transport and virtual switches for propagation. In OpenVPN and SoftEther, the central server sends broadcast traffic to all connected clients. tinc maintains its own spanning tree routing structure that is used to send messages to all connected clients during broadcasts, and ZeroTier uses sender-side replication and internal multicast groups. All solutions that support broadcast traffic have also some kind of loop prevention.

3.2 Performance

In order to assess the performance of the VPN candidates, we measured throughput and latency between each VPN client and server with the standard network utilities ping and iperf3 in two different deployments, with two virtual machines

	IPsec	OpenVPN	SoftEther	tinc	WireGuard	ZeroTier
Forwarding						
Operation Mode	L3	L2/L3	L2/3	L2/L3	L3	L2
Mesh topology	✓	✗	✗	✓	✓	✓
Link deactivation	✓	✗	✗	✗	✓	✗
Broadcast replication	None	Central	Central	Tree	None	Sender
Performance						
Multi-core utilization	✗[1]	✗	✗[1]	✗	✓	✓[1]
Cloud deployment[2] (Download: 948 Mb/s; Upload: 882 Mb/s; RTT Latency: 3.1 ms)						
Download (Mb/s)	347	139	5[3]	138	439	6[4]
Upload (Mb/s)	260	111	146	132	489	5[4]
Additional RTT (ms)	0.2	0.5	0.8	0.3	0.2	0.5
Testbed deployment[2] (Throughput: 4.370 Mb/s; RTT Latency: 1.6 ms)						
Throughput (Mb/s)	211	76	131	83	577	155
Additional RTT (ms)	0.2	1.7	2.5	1.3	1.6	1.5
Security						
No cipher agility	✗	✗	✗	✗	✓	✓
Formally verified	✗	✗	✗	✗	✓	✗
No known weaknesses	✓	✓	✓	✗	✓	✓
No single point of failure	✓	✗	✗	✓	✓	✗
Configuration						
Central configuration	✗	✓	✓	✗	✗	✓
Central services	None	PKI	PKI	None	None	Discovery
Multiplexing						
Protocol[5]	GRE	VLAN	VLAN	VLAN	GRE	Native
Maximum number of networks	2^{32}	2^{12}	2^{12}	2^{12}	2^{32}	$> 2^{24}$
Additional packet size (Bytes)	4	4	4	4	4	0

[1] This is not documented and only based on observations during our benchmark
[2] Instead of distinct values for download/upload, testbed measurements only report one throughput value as the links were basically symmetrical
[3] Inconclusive due to a suspected implementation bug
[4] Inconclusive due to a known implementation bug
[5] For protocols without built-in network multiplexing support, we assume the most-efficient extension, depending on their operation mode

Table 1. VPN Protocol Comparison

each. In the first case we set them up on two different cloud platforms, and in the other case on dedicated hosts in our local testbed. For each deployment, we did some reference measurements to determine the baseline throughput and latency (see Table 1). All measurements were executed for 60 seconds and an average value was calculated over the whole time period.

The virtual machines of the cloud deployment were equipped with one virtual CPU and 2 GB of RAM each. WireGuard with a throughput of 439 Mb/s (down) and 489 Mb/s (up) as well as a round-trip overhead of only 0.2 ms is by far the best performing VPN solution in this scenario. IPsec comes in on the second place with only half of WireGuard's performance. OpenVPN, SoftEther and tinc rank on a comparable level with only a quarter of WireGuard's performance. ZeroTier turned out to be extremely slow while causing high CPU load on the sending host. This turned out to be a known performance issue with single CPU machines in current versions of ZeroTier [7]. A similar problem also manifested in the download measurements of SoftEther. Here, we were not able to find additional information or the limiting factor.

In the testbed deployment we used virtual machines with two CPU cores and 1 GB of RAM. This remedied the performance issues with ZeroTier and SoftEther. As the link quality in this setup was symmetrical, we only report one value for throughput. The trends from the previous results are also reflected in these measurements. WireGuard once again performs best, with 577 Mb/s. IPsec follows with 211 Mb/s and after that we can find two clusters with around 140 Mb/s (ZeroTier and SoftEther) and 80 Mb/s (tinc and OpenVPN).

Driven by the issues with the single core CPU, we also investigated if the tested VPN implementations make use of multi-core architectures. We could observe performance gains only with WireGuard and ZeroTier. For WireGuard, OpenVPN and tinc we found documentation that confirms our findings, for the others we can only rely on the observations during our measurements. Parallelizing strongswan (Debian's default IPsec implementation) is possible with the kernel module pcrypt, but we did not use any customizations during our tests.

3.3 Security

Our general security requirements – authentication of peers as well as integrity and confidentiality of transported data – can be provided by all tunnel protocols. Cipher agility denotes the negotiation of cipher algorithms between peers during the initial handshake. No cipher agility results in a simpler and more secure protocol, if the protocol is actively maintained. In case ciphers are broken, the whole software must be updated, but cipher configuration updates are at least just as burdensome. All VPNs support asymmetric authentication. IPsec, WireGuard, tinc and ZeroTier use asymmetric key pairs without certificates, whereas OpenVPN and SoftEther utilize a Public Key Infrastructure (PKI). In addition, IPsec, OpenVPN and SoftEther allow further authentication methods like pre-shared keys, passwords as well as external authentication protocols and frameworks. Regarding known weaknesses, tinc lists some issues allowing man-in-the-middle and chosen plaintext attacks, which were fixed in the preliminary

release 1.1 [22]. However, this release has not yet been made stable since 2018. If we investigate possible failure points, it becomes evident that OpenVPN and SoftEther offer single points of failures due to their centralized structure. The same applies to their PKI, if certificates revocations are used, and to ZeroTier's discovery service. We found protocol verifications or cryptographic evaluations only for WireGuard and IPsec, with WireGuard being the only formally verified protocol. But even with those analyses, the varying encryption and authentication approaches make it hard to absolutely assess the security of the solutions.

3.4 Configuration Complexity

As configuration can be automated and its complexity is a rather subjective measure, we only assess the number of configuration updates required whenever peers are added or removed dynamically. Hereby, we can put all candidates in one of two groups. OpenVPN, SoftEther and ZeroTier use a centrally configured star topology and hereby need to connect to one host for reconfiguration. IPsec, tinc and WireGuard, on the other hand, can run on a mesh topology that requires all other peers to be contacted in order to advertise the updated configuration. In addition to that, ZeroTier requires an additional proprietary discovery service, which delivers the initial configuration or points to custom configuration servers. Also, OpenVPN and SoftEther suffer from additional configuration efforts with a PKI, if asymmetric authentication is assumed. OpenVPN and SoftEther could also use a custom authentication mechanism, but this would require additional implementation efforts. Using a pre-shared key is also no option in a star topology with dynamically joining and leaving nodes, as removed nodes could reconnect with the already known secret.

3.5 Network Multiplexing

To support multiple individual overlay networks, the VPN technology should support network multiplexing. For IPsec and WireGuard, GRE is able to create up to 2^{32} distinct tunnels between each host pair utilizing the GRE Key Extension, which adds an overhead of 4 bytes to each packet. OpenVPN, SoftEther and tinc have no built-in support for network multiplexing, but the application of VLAN offers 2^{12} virtual networks over any Ethernet network with a 4 byte overhead per packet. It would also be possible to use a distinct VPN with a unique UDP port for each individual network resulting in 2^{16} possible networks and no per-packet overhead. However, this approach would not scale very well as the configuration complexity would also increase with number of networks. In a similar way, ZeroTier's built-in network multiplexing supports up to 2^{24} networks without packet or large configuration overhead. In summary, all protocols support network multiplexing in some way, but with regards to the number of possible networks and their transport and configuration overhead, ZeroTier is the optimal approach, followed by IPsec and WireGuard using GRE, with a slightly larger number of available networks.

3.6 Conclusion

In summary, we see WireGuard in combination with GRE as the most promising VPN tunnel protocol. Mainly because of its performance results, but also because of its simple and secure design, which has been formally verified. We value the the higher configuration efforts when peers are added or removed as well as the missing broadcast replication only as minor disadvantages that can be tolerated.

4 Approach

Figure 1 illustrates a high-level overview of our NetFed architecture for the automated federation of isolated network segments, showing two sample federated networks spanning three sites. Networks are either bridged Ethernet or routed dual-stack IPv4/6 networks, consisting of multiple virtual or physical network segments inside a company site or cloud platform. The segments are connected through federation agents (or peers), which run tunnels that forward the traffic isolated from each other. NetFed knows two types of tunnels. Peer tunnels, which connect two federation agents site-to-site, and neighbor tunnels, which connect a federation agent with a host (or neighbor) in the same segment point-to-site.

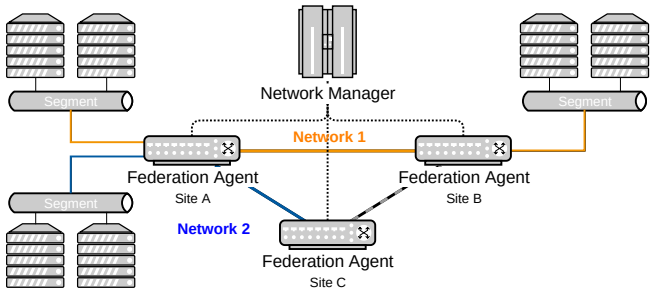


Fig. 1. High-level overview of two sample federated networks spanning three sites

At least one federation agent is placed in each site and all agents are connected with each other in a mesh topology. Additional agents could be utilized to balance the load or isolate certain networks. Links can be deactivated in order to ensure loop prevention and efficient routing. Federation agents run virtual switches and routers as GRE and WireGuard tunnel endpoints. A dynamic configuration interface is exposed to the network manager. This central control instance knows all federation agents and initializes the tunnels during startup based on an external description or dynamically adapts the tunnel configurations during runtime. The deployment of the federation agents is the responsibility of the cloud orchestrator and not covered in this paper. We followed an agent-based federation approach, as it constitutes a well-established approach and guarantees full control over the network segments without any vendor dependencies or

restrictions. Thus, our solution can be deployed to arbitrary cloud platforms and computing environments.

4.1 Operation Modes

NetFed offers two operation modes – bridged and routed – to provide layer 2 and layer 3 federation. In bridged mode, the federation agents act as virtual switches in order to connect the isolated Ethernet fragments. This behavior is presented in Figure 2 with Network 0 between Site A and Site B. This setup offers the highest flexibility and transparency, as it forwards any traffic and requires no further host configuration.

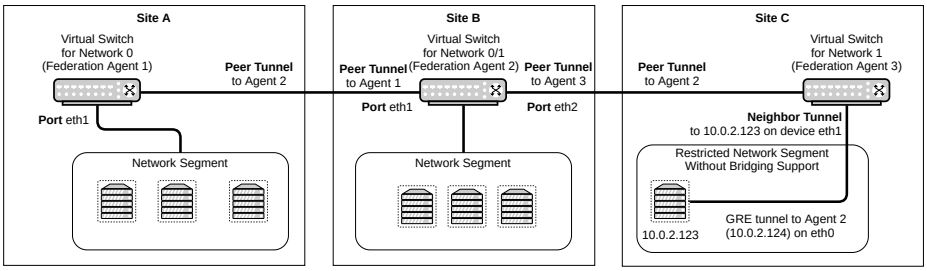


Fig. 2. Bridged mode: Peer tunnels between virtual switches connect the reachable hosts of network segments in Network 0; additional neighbor tunnels inside the local segment of Site C connect hosts to the virtual switch directly

However, some cloud platforms do not support BUM packets or do not forward layer 2 traffic originating from unknown/external MAC addresses. To cope with such platforms NetFed offers neighboring tunnels. One has to sacrifice the transparency property, since bridged mode with neighbor tunnels requires explicit host configuration. In this case, federation agents establish GRE tunnels to all hosts in the local segment, as illustrated in Figure 2 with Network 1 between Site B and Site C.

The second mode NetFed can be used with is the routed mode. Here, the federation agents act as virtual routers for their network segment, as shown in Figure 3. The data is forwarded based on configured routes. The routes for the remote networks must be enforced by the segment's default gateway or pushed to the hosts with DHCP.

4.2 Internal Networking

To achieve this flexibility, NetFed leverages the network protocol stack shown in Figure 4(a) on its federation agents. At the bottom WireGuard serves as the VPN foundation. Therefore, every agent must be reachable from the Internet on WireGuard's de facto standard UDP port 51820 – directly or also indirectly

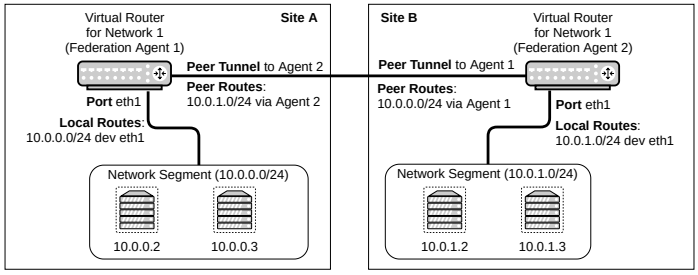
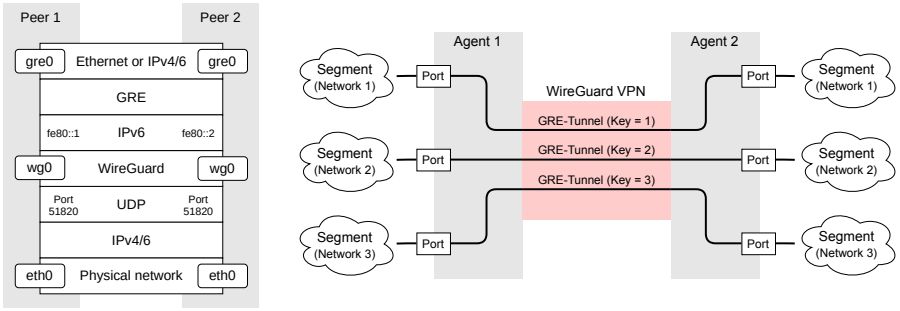


Fig. 3. Routed mode: Peer tunnels between virtual routers

through some NAT layer. To prevent address collisions with existing private IPv4 networks within some agent's environment, NetFed assigns IPv6's link-local addresses inside the WireGuard network, which can freely be chosen from the fe80::/64 address block and can carry an additional zone identifier [6]. This way, each federation agent gets assigned an IPv6 address with its numerical identifier mapped to the last 64 bits (e.g. agent with identifier 1 will be assigned fe80::1). With multiple ports, the additional identifier (e.g. fe80::1%eth0) ensures collision resistance.



(a) Protocol Stack (b) GRE multiplexing

Fig. 4. Internal networking: GRE over WireGuard

On top of a WireGuard VPN tunnel, multiple GRE tunnels can be established between each pair of federation agents. Figure 4(b) shows this behavior for three federated networks. Network multiplexing is achieved by simply putting the network's identifier into the GRE tunnel's key field [18]. For every federated network, a distinct virtual bridge device is created on each federation agent. By establishing selected tunnels between the agents, the network manager is able to shape the network topology based on external information such as link costs or traffic restrictions. Establishing tunnels between all federation agents forms a

full mesh. In this case, additional protocols like the Spanning Tree Protocol [8] or Open Shortest Part First [18] can be used to prevent loops and optimize routing.

4.3 Deployment

In order to deploy NetFed, an external cloud orchestrator needs to provision the federation agents. This can be done by installing the federation agent software stack on the already existing gateways or by deploying the federation as virtual appliance, but we do not restrict NetFed to any particular infrastructure as code (IaC) technology. For every federation agent, its public IP address or hostname must be made available to the network manager. With this information at hand, the manager first establishes WireGuard VPN connections between the federation agents and then creates the peer tunnels to deploy the federated networks. The network manager only needs to be online during the initial deployment or whenever reconfigurations are needed. Future versions could replicate the federation agents to improve their reliability and scalability. Additional services like monitoring or policy enforcement could be integrated into the central network manager, which then should also be replicated to avoid a single point of failure.

5 Evaluation

We provide a prototype implementation for the federation agent, which is written in Python and implemented as command line agent. In the following we present our evaluation results in terms of functionality, security and performance.

5.1 Functionality

We tested the general functionality of our prototype implementation in an extended setup connecting multiple hosts in four different sites. With AWS and OpenNebula, we used a public and a private cloud, and further integrated two local network segments, one physical LAN and a virtualized network. NetFed is able to federate such networks in both modes. The bridged mode must make use of neighbor tunnels, since AWS and OpenNebula will drop packets with unknown source MAC addresses, as already described above.

However, during the functionality tests some issues with the maximum transmission unit (MTU) size occurred. Principally, the MTU sizes of all connected segments should match, and also be smaller than the WireGuard MTU without the GRE overhead, as packages exceeding this size will be silently dropped. Running Ethernet over GRE in our neighbor tunnels adds a combined overhead of 70 bytes if WireGuard runs on top of IPv6 (14 (Ethernet) + 8 (GRE) + 48 (IPv6, with Destination Options Header)) or 42 bytes in case of IPv4 (14 (Ethernet) + 8 (GRE) + 20 (IPv4)). Therefore, the MTU size of bridged networks must be set to the minimum of: the WireGuard network's MTU minus 70; the MTU sizes of the connected segments; and the MTU sizes of all connected segments running neighbor tunnels minus 70 (for IPv6 tunnels) or minus 42 (for IPv4 tunnels).

With 1500 as the default Ethernet MTU, 1350 turned out to be a good choice to prevent fragmenting. This configuration option must be set on every host, what affects the transparency property in a negative way. It would be also possible to increase the WireGuard MTU on the agents only, but this would mean that WireGuard packets would be fragmented during their transport.

5.2 Security

The security guarantees of physical local networks should also hold for our federated networks. Thus, we performed a security analysis with attack tree modeling, which is not fully included due to length restrictions. The analysis is based on the following assumptions: (i) there is already an established trust relationship between the network manager and all federation agents that allows the manager to access the administrative interface of the agents via SSH; (ii) the network manager is never compromised, as its administrative access to the agents would compromise all federated networks; (iii) the underlying software stack is free of implementation bugs and provides it specified security properties; and (iv) the utilized strong cryptography is infeasible to break.

Following those assumptions, the integrity and confidentiality of the communication sent over federated networks is ensured through the authentication and encryption of the WireGuard VPN. The agent configuration in transit is protected in the same way by SSH. If an attacker gains control over a host in a network segment, either by compromising an existing host or by introducing a new host with the assistance of a compromised cloud platform, it can obviously read the traffic directed to that host and send traffic into all connected segments of that host's networks. Access to other federated networks, however, remains restricted. This corresponds to the behavior of physical networks.

Consequently, federation agents might constitute a more valuable attack target to attackers. But, introducing a new federation agent is a futile undertaking, since WireGuard requires an established trust relationship between all peers, in which all other peers need to learn the public key of a newly introduced peer. This trust can only be established by the network manager, which we assume to be secure. If an attacker compromises an existing federation agent, it can read and modify the traffic of all segments of the networks, to which this agent is connected, as well as introduce new traffic. Re-routing traffic by impersonating existing peers is, however, still not possible on compromised hosts, due to WireGuard's CryptoKey Routing [2].

All in all, it can be concluded that the security of NetFed heavily depends on the underlying components. We cannot fully prevent security breaches carried out by attackers that maliciously gained administrative access to cloud platforms or access to individual hosts, agents and their private keys. However, NetFed prevents attackers, without access to any host or agent in a federated network, from reading, manipulating or injecting packets in that network. And, attackers, with access to a host or agent, from reading, manipulating or injecting packets in other federated networks.

5.3 Performance

For the performance evaluation, we set up a federated network consisting of two segments with four hosts per segment. One segment was located in our private OpenNebula cloud (Passau, Germany) and the other segment was hosted in the eu-central-1 region of the public AWS cloud (Frankfurt, Germany). We run the tests with a bridged network using neighbor tunnels and a routed network.

Traffic Overhead To determine the actual traffic overhead we sent one gigabyte of random data from one host in the one cloud to another host in the other cloud using the netcat utility. We measured the exchanged traffic on its way between the different hosts and responsible federation agents, as illustrated in Figure 5. The results are shown in Table 2 with the TCP traffic between the two hosts as reference measurement. The main difference between the operation modes is represented by the 2.5% traffic overhead, which the neighbor tunnels add in bridged networks on the way from the host to the agent. The remaining overheads behave similar. GRE increases the data exchanged between the agents by 3.9% in the bridged setup and 3.2% in the routed set up. The IPv6-based WireGuard tunnel elevates those numbers to 17.0% and 15.1%

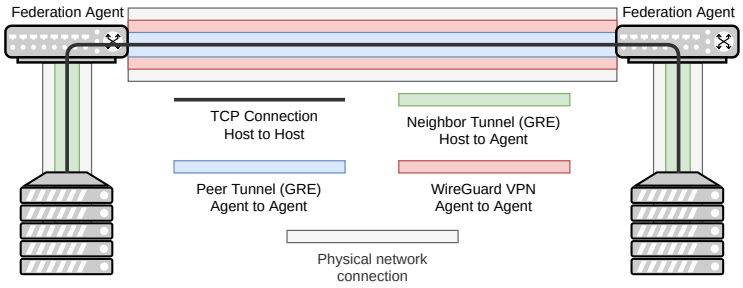


Fig. 5. Overview of the different types of nested tunnels in a federated network.

	Bridged Setup	Routed Setup
Payload	1024 MiB	1024 MiB
TCP Host-to-Host	1060 MiB (100%)	1060 MiB (100%)
All Host-to-Agent	1087 MiB (+2.5%)	1060 MiB (+0.0%)
GRE Agent-to-Agent	1102 MiB (+3.9%)	1094 MiB (+3.2%)
All Agent-to-Agent	1240 MiB (+17.0%)	1220 MiB (+15.1%)

Table 2. Comparison of generated traffic when sending a fixed amount of random data over TCP. Relative deviations are in comparison to the TCP Host-to-Host traffic.

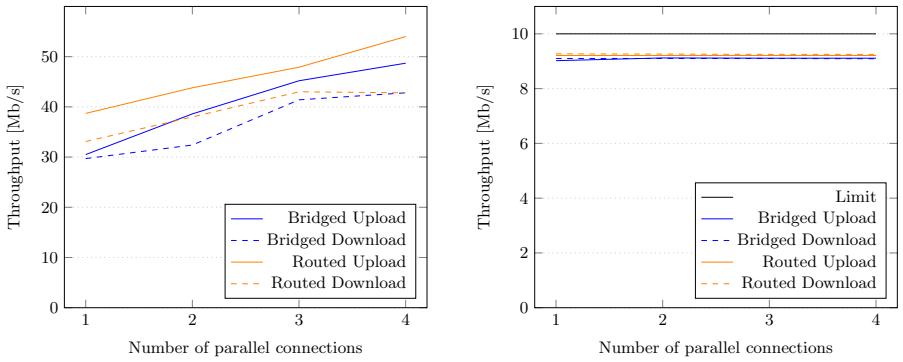
Latency We also investigated the influence of our tunnel network on latency using the same setups. For this purpose, we used ping to measure the round-trip time between two hosts in different clouds. We performed the measurements either with or without additional traffic, where one or multiple host pairs exchange data at a fixed rate or even unrestricted. We included once again a reference measurement without any federation. The results are shown in Table 3, from which we can see that the routed traffic is generally forwarded a little faster. This can be explained by the additional processing delay introduced by the GRE overlay. Apart from that, it also becomes clear that additional traffic on the network has a negative influence on the overall latency, which is most likely be caused by additional queuing delays. However, we cannot rule out minor deviations due to the external conditions the traffic experiences over the Internet.

	Bridged Setup	Routed Setup
Without tunnel	13.2 ± 2.8 ms	12.9 ± 3.7 ms
0 Mb/s	13.2 ± 1.6 ms	13.0 ± 2.6 ms
8 Mb/s	13.9 ± 3.1 ms	16.0 ± 8.6 ms
16 Mb/s	13.9 ± 2.6 ms	13.7 ± 4.8 ms
Full load (1 connection)	14.6 ± 3.4 ms	14.2 ± 4.8 ms
Full load (3 connections)	13.5 ± 2.2 ms	14.4 ± 6.2 ms

Table 3. Latency (and standard deviation) between two hosts in different segments using a direct connection vs. a network federation setup with different levels of load.

Throughput We measured the total throughput of multiple host pairs exchanging data between two sites to determine throughput and the overhead of multiple connections. Measurements were done in both directions, with the OpenNebula hosts as clients and the AWS hosts as servers, so that Figure 6 consequently represents the throughput as upload and download. Figure 6(a) shows that the overall throughput actually rises with more parallel connections. This counterintuitive behavior can be explained with the fairness property of TCP, which will be activated when multiple connections share a common bottleneck. In this case the congestion control algorithm tries to distribute the available bandwidth in fair shares over the competing connections, which results in a larger combined bandwidth than a single collection could obtain. In order to assess the actual overhead, we restricted the outgoing throughput of both agents to 10 Mb/s in order to create a non-shared bottleneck, as Figure 6(b) illustrates. In bridged mode, we measured a throughput of 9.11 Mb/s (up) and 9.10 Mb/s (down), whereas in routed mode the throughput reached 9.21 Mb/s (up) and 9.26 Mb/s (down), regardless of the number of parallel connections. Hereby, the federated networks showed a relative network-bound overhead of about 9% for the bridged setup and 8% for the routed setup. It has to be noted that federation agents can

in fact become bottlenecks, which is why the expected network load should be reflected in the agents' hardware configuration or the networks' topology.



(a) Unthrottled connection (b) Limited to 10 Mb/s

Fig. 6. Total throughput by number of parallel connections over a single tunnel

Neighbor Tunnels So far we only investigated the communication between different segments. However, if neighbor tunnels are used, there is also an overhead on in-segment communication. Therefore, we measured the latency and throughput between a pair of hosts belonging to the same segment in our OpenNebula environment – with and without neighbor tunnels. Without, we measured a throughput of 938 Mb/s and a round-trip latency of 0.50 ms. With tunnels, those values degraded to 602 Mb/s for throughput and 0.81 ms for latency. This local overhead must be considered in bridged inter-cloud operations.

6 Conclusion

In this work, we presented a fast and secure network federation approach that enables flexible and multi-cloud operations. From our analysis of existing tunneling protocols we selected WireGuard as the most secure and fastest-performing technology. The light-weight NetFed architecture is able to support a wide range of use case scenarios, while preserving network functionality, configuration transparency and cloud platform support. Our prototype implementation shows its practicability and the performance evaluation proves its efficiency. We still see room for improvements in various areas, such as cloud orchestration integration or an extended configuration management, but we leave this open as future work.

Acknowledgements

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 268730775 (OptSCORE).

References

1. Al-Shabibi, A., De Leenheer, M., Gerola, M., Koshibe, A., Parulkar, G., Salvadori, E., Snow, B.: OpenVirteX: Make your virtual SDNs programmable. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. pp. 25–30. HotSDN '14, Association for Computing Machinery, New York, NY, USA (Aug 2014). https://doi.org/10.1145/2620728.2620741
2. Donenfeld, J.: WireGuard: Next Generation Kernel Network Tunnel. In: Network and Distributed System Security Symposium (NDSS) (Feb 2017). https://doi.org/10.14722/ndss.2017.23160
3. Gaul, C., Körner, M., Kao, O.: Design and Implementation of a Cloud-Federation Agent for Software Defined Networking. In: 2015 IEEE International Conference on Cloud Engineering. pp. 323–328 (Mar 2015). https://doi.org/10.1109/IC2E.2015.58
4. Goethals, T., Kerkhove, D., Van Hoye, L., Sebrechts, M., De Turck, F., Volckaert, B.: FUSE: A Microservice Approach to Cross-domain Federation using Docker Containers:. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science (May 2019)
5. Google Cloud Platform: VPC network overview, `https://cloud.google.com/vpc/docs/vpc`, last accessed on 2021-02-18
6. Hinden, R.M., Haberman, B.: Unique Local IPv6 Unicast Addresses. RFC 4193 (October 2005), `http://www.rfc-editor.org/rfc/rfc4193.txt`
7. Holden, J.: cpu pegged at 100% (November 2019), `https://github.com/zerotier/ZeroTierOne/issues/1079`, last accessed on 2021-02-18
8. Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges. IEEE Std 802.1D-2004 pp. 137–179 (June 2004). https://doi.org/10.1109/IEEESTD.2004.94569
9. Kimmerlin, M., Hasselmeyer, P., Heikkilä, S., Plauth, M., Parol, P., Sarolahti, P.: Network expansion in OpenStack cloud federations. In: 2017 European Conference on Networks and Communications (EuCNC). pp. 1–5 (Jun 2017). https://doi.org/10.1109/EuCNC.2017.7980655
10. Levin, A., Barabash, K., Ben-Itzhak, Y., Guenender, S., Schour, L.: Networking Architecture for Seamless Cloud Interoperability. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 1021–1024 (Jun 2015). https://doi.org/10.1109/CLOUD.2015.141
11. Mansouri, Y., Prokhorenko, V., Babar, M.A.: An Automated Implementation of Hybrid Cloud for Performance Evaluation of Distributed Databases. Journal of Network and Computer Applications p. 102740 (October 2020). https://doi.org/10.1016/j.jnca.2020.102740
12. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review (2), 69–74 (Mar 2008). https://doi.org/10.1145/1355734.1355746
13. Mechtri, M., Zeghlache, D., Zekri, E., Marshall, I.J.: Inter and intra Cloud Networking Gateway as a service. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet). pp. 156–163 (Nov 2013). https://doi.org/10.1109/CloudNet.2013.6710570
14. Medved, J., Varga, R., Tkacik, A., Gray, K.: OpenDaylight: Towards a Model-Driven SDN Controller architecture. In: Proceeding of IEEE International Sym-

posium on a World of Wireless, Mobile and Multimedia Networks 2014. pp. 1–6 (June 2014). https://doi.org/10.1109/WoWMoM.2014.6918985
15. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Cross-Site Virtual Network in Cloud and Fog Computing. IEEE Cloud Computing (2), 46–53 (March 2017). https://doi.org/10.1109/MCC.2017.28
16. Moreno-Vozmediano, R., Huedo, E., Llorente, I.M., Montero, R.S., Massonet, P., Villari, M., Merlino, G., Celesti, A., Levin, A., Schour, L., Vázquez, C., Melis, J., Spahr, S., Whigham, D.: BEACON: A Cloud Network Federation Framework. In: Advances in Service-Oriented and Cloud Computing. pp. 325–337. Communications in Computer and Information Science, Springer International Publishing, Cham (April 2016). https://doi.org/10.1007/978-3-319-33313-7_25
17. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Implementation and Provisioning of Federated Networks in Hybrid Clouds. Journal of Grid Computing (2), 141–160 (June 2017). https://doi.org/10.1007/s10723-017-9395-1
18. Moy, J.: OSPF Version 2. RFC 2328 (April 1998), `http://www.rfc-editor.org/rfc/rfc2328.txt`
19. Nadjaran Toosi, A., Buyya, R.: Virtual Networking with Azure for Hybrid Cloud Computing in Aneka. In: Research Advances in Cloud Computing, pp. 93–114. Springer, Singapore (December 2017). https://doi.org/10.1007/978-981-10-5026-8_5
20. Natarajan, S., Ramaiah, A., Mathen, M.: A Software defined Cloud-Gateway automation system using OpenFlow. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet). pp. 219–226 (November 2013). https://doi.org/10.1109/CloudNet.2013.6710582
21. Pepelnjak, I.: AWS Networking 101 (June 2020), `https://blog.ipspace.net/2020/05/aws-networking-101.html`, last accessed on 2021-02-18
22. Sliepen, G.: Tinc version 1.0.35 and 1.1pre17 released (October 2018), `https://www.tinc-vpn.org/pipermail/tinc/2018-October/005311.html`, last accessed on 2021-04-23
23. Tai, S., Klems, M., Lenk, A., Kunze, M., Bermbach, D., Kurze, T.: Cloud Federation. In: Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011). IARIA (September 2011)
24. Wood, T., Gerber, A., Ramakrishnan, K.K., Shenoy, P., Van der Merwe, J.: The Case for Enterprise-Ready Virtual Private Clouds. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing. HotCloud'09, USENIX Association, USA (June 2009)
25. Xu, Z., Di, S., Zhang, W., Cheng, L., Wang, C.: WAVNet: Wide-Area Network Virtualization Technique for Virtual Private Cloud. In: 2011 International Conference on Parallel Processing. pp. 285–294 (September 2011). https://doi.org/10.1109/ICPP.2011.90