Designing a Transparent Network Fabric for Cloud-Native Federated Services

Designing a Transparent Network Fabric for Cloud-Native Federated Services

Davide Miola
Dept. of Computer and Control Engineering
Politecnico di Torino
Turin, Italy
davide.miola@polito.it

Francesco Cheinasso
ArubaKube
Turin, Italy
cheinasso.francesco@gmail.com

Fulvio Risso
Dept. of Computer and Control Engineering
Politecnico di Torino
Turin, Italy
fulvio.risso@polito.it

***Abstract*—Computing federation represents one of the most influential trends in today's cloud discussions, thanks in part to the continuous increase in computing needs. Indeed, federating clusters can induce several advantages over a flat expansion of one's cloud resources, including the potential for reduced operational costs, access to clean energy, and improved latency for end users. These compelling arguments have led to a multitude of inter-cloud mesh solutions being designed around the need to interconnect cooperating Kubernetes clusters through a fabric that allows the seamless offload of workloads and services. This paper serves as an objective comparison of the available solutions, with a specific focus on the Liqo network fabric and its future developments.**

***Index Terms*—Cloud computing, Kubernetes, Cloud federation, Multi-cluster, Hybrid cloud**

I. Introduction

Kubernetes has effectively matured as the undisputed standard for container orchestration in computing clusters, as its adoption continues to accelerate in cloud-native environments [1]. Within this landscape, many organizations have expressed interest in deploying multi-cluster architectures as a way to meet increasing scalability demands, to improve resilience and availability through geographic distribution, decrease end-to-end latency via seamless hybrid-cloud setups, and more. However, handling the networking layer between multiple and independent Kubernetes clusters presents fundamental challenges, such as the necessity to transparently support heterogeneous environments and topologies, and to accommodate potentially incoherent IP addressing strategies.

In this paper, we first (Section II) formalize the requirements that a multi-cluster network fabric must have in order to serve as a consistent extension of an individual Kubernetes cluster; then, in Section III, a comprehensive overview and comparison of the available projects attempting to fulfill this task is given, outlining shortcomings and limitations of the current pen-source landscape. Focusing on Liqo specifically, the remainder of this paper will (*i*) explain the design of its network model in detail (Section IV), (*ii*) describe some of the persisting limitations of this design, as well as the ongoing efforts being made to mitigate them (Section V). Finally, the paper is concluded in Section VI.

II. Requirements of Cross-Cluster Network Fabrics

To qualify as a true *cluster mesh* fabric for Kubernetes, network plugins must meet a set of key requirements that enable seamless, secure, and scalable inter-cluster communication. We formalized these features into the following list:

1) **Cryptographically secure and extensible interconnect.** In scenarios where participating clusters are not connected through private networks, the mesh must ensure encryption of all inter-cluster traffic. Additionally, its design should be extensible, allowing operators to select among multiple interconnect backends (e.g., IPsec, WireGuard) according to specific deployment needs, with the option to disable encryption when deemed unnecessary.
2) **Direct Pod-to-Pod communication across cluster boundaries.** The fabric must enable transparent IP-layer connectivity between Pods across all clusters in the federation, such that they can communicate as if they were co-located within a single cluster.
3) **"Global" services.** The system should provide mechanisms to expose services defined in one cluster to other clusters in the federation, enabling remote Pods to transparently consume them.
4) **Heterogeneous service endpoints.** Globally exposed services must support endpoints distributed across multiple clusters, enabling load balancing and high availability at a multi-cluster scale.
5) **Compatibility with heterogeneous CNIs and cluster configurations.** Given that federations may span independent administrative domains with diverse networking stacks, the mesh fabric must remain agnostic to the specific *Container Network Interface* (CNI) implementations and cluster configurations in use.
6) **Support for overlapping CIDRs.** Address space collisions across clusters represent a possible concern in real-world federations, as organizations reuse default settings. The network fabric must be capable of detecting such overlaps and resolving them through automated address translation or remapping mechanisms.

979-8-3195-3958-8/26/$31.00

TABLE I
SYSTEMATIC COMPARISON OF THE PRESENTED CLUSTER MESH SOLUTIONS W.R.T. SECTION II REQUIREMENTS.

Metric	Submariner	Skupper	Liqo	Cilium ClusterMesh	Calico ClusterMesh	Istio Multicluster	Linkerd Multicluster
Architecture	Overlay w/ Gateway	L7 VAN	Overlay w/ Gateway	Overlay or Node-to-Node	Overlay or Node-to-Node	Proxy w/ overlay or Node-to-Node	Sidecar w/ overlay or Node-to-Node
Interconnect technology	IPsec, WireGuard, VXLAN, extensible	mTLS	WireGuard, extensible	IPsec, WireGuard	WireGuard	mTLS	mTLS
Generic IP routing between clusters	Yes	Pod-to-SVC, TCP only	Yes	Yes	Yes	Pod-to-SVC, TCP only	Pod-to-SVC, TCP only
Global services	Yes, automatic mirroring	Yes, automatic mirroring	Yes, automatic mirroring	Yes, manual replication	Yes, manual replication	Yes, manual replication	Yes, automatic mirroring
Heterogeneous service endpoints	Yes	Yes	Yes	Yes	Yes	Yes	Only on flat networks
Agnostic to CNI and cluster configuration	Yes, Cilium not officially supported	Yes	Yes	Cilium CNI only	Calico enterprise CNI only	Requires Istio service mesh	Requires Linkerd service mesh
Overlapping CIDR support	Yes, with *Globalnet* addressing	Yes	Yes, CIDR remapping	No	No	Only on hierarchical networks	Only on hierarchical networks

III. ONTOLOGY OF AVAILABLE SOLUTIONS

Several projects have been developed to provide the aforementioned features. Some of these are specialized, ad-hoc produces that sit on top of the Kubernetes CNI plugin (thus generally being agnostic to it) to offer inter-cluster connectivity functionalities. Others, instead, qualify as extensions of lower-level solutions such as CNI and Service Mesh plugins. We describe both approaches with specific examples in the following sub-sections.

A. Specialized projects

Submariner [2] is a sandbox project under the Cloud Native Computing Foundation (CNCF) that provides multi-cluster networking capabilities for Kubernetes. It adopts a broker-based architecture, in which a central cluster – which must be reachable by all participating clusters – coordinates the federation process. The data plane supports CNI-agnostic inter-cluster Pod communication and global service discovery through the *Lighthouse* component, which implements the *Kubernetes Multi-Cluster Services* (MCS) API [3]. Cross-cluster communication is mediated by leader-elected gateway Nodes, which encapsulate traffic using pluggable "cable" drivers. Both IPsec and WireGuard [4] are supported for securing in-transit traffic, while a plaintext VXLAN-based option is available for scenarios where encryption is unnecessary. Submariner also provides optional support for overlapping Pod and Service CIDRs via the *Globalnet* component, which assigns globally unique, non-overlapping IP addresses to Pods, Services and Nodes. When available, these addresses are automatically leveraged by the network fabric to ensure correct routing across the mesh.

Skupper [5] brings the concept of Virtual Application Networks (VANs) to multi-cluster environments by establishing a Layer-7 overlay network over individual Kubernetes namespaces. This is mediated by their application routers, which effectively serve as TCP proxies between the shared services. Skupper ensures data confidentiality and authentication thanks to the use of mTLS encryption between the routers, though the L7 nature of the project means that it cannot support generic IP connectivity between Pods – it does not, for example, natively handle UDP –, while rather focusing on exposing global services across clusters.

Liqo [6], [7] is a complete Kubernetes cluster mesh solution, integrating both control and data plane federation capabilities. Cluster pairs collaborating through a Liqo-powered mesh establish a unidirectional peering relationship that distinguishes the two into provider and consumer roles, allowing for fine-grained control of resource allocation in the computing continuum. Through the namespace extension feature, Liqo allows select namespaces to be shared between peering clusters, enabling their resources – including pods and services – to be offloaded by the consumer to the provider. The Liqo network fabric supports this functionality via an overlay WireGuard-based VPN that interconnects peered clusters, ensuring data confidentiality and enabling seamless L3 routability between Pods and Services across interconnected clusters. A more in-depth overview of the Liqo network fabric's architecture is presented in Section IV.

B. CNI and Service Mesh extensions

Cilium Cluster Mesh [8] enhances the popular Cilium CNI plugin for Kubernetes with inter-cluster meshing capabilities. By supporting both tunneled overlay networks and direct routing between the cluster' nodes, it can provide high levels of flexibility and performance while retaining policy enforcement capabilities thanks to the deep integration with the in-

cluster eBPF-based backend, at the cost of requiring that all federating clusters run the associated Cilium CNI plugin. It offers seamless inter-cluster routing and global service load-balancing, though it remains incompatible with overlapping address ranges at this time.

Similarly to Isovalent's offering, the *Calico Cluster Mesh* [9] offers most of the same features to users of the enterprise versions of the Calico CNI plugin.

Finally, the *Istio* [10] and *Linkerd* [11] *Multicluster* products represent two of the most widely adopted service mesh-based solutions for enabling cluster federation in Kubernetes environments. Although these approaches do not require a specific CNI plugin, their use necessitates full integration with their respective service mesh platforms, making them impractical for federating clusters managed by different organizations. Both systems are fundamentally built on a sidecar architecture, where each application Pod is injected with a dedicated proxy process (Istio additionally supports an optional "*Ambient Mode*", where the proxy is deployed on a one-per-Node basis for performance and scalability reasons, though overall functionality is unchanged). This sidecar is responsible for delivering core service mesh functionalities, such as traffic management, observability, and security, to cloud-native workloads. To insert the proxy in the application's network data path, a set of Netfilter rules is automatically configured within the Pod's network namespace, ensuring that all ingress and egress traffic is routed through it. When Multicluster extensions are enabled, the proxy becomes aware of off-cluster Services, thus extending the communication and operational scope of microservices beyond the boundaries of a single cluster.

C. Comparison

Table I compares these multi-cluster projects on the grounds of the stated requirements regarding security, flexibility, and overall functionality. Although all entries have basic support for secure multi-cluster service extension with heterogeneous endpoints, some require manual replication of the Service definition across participating clusters – on the principle of *Namespace Sameness* – in order to effectively share it over the mesh, while others facilitate this by automatically mirroring specifically labeled or otherwise annotated Services. Moreover, L7-based solutions such as both Service Mesh extensions and Skupper, fall short of providing generic IP routability across clusters due to their TCP-centric design and focus on Service extension rather than Pod-to-Pod communication across the federation. Additionally, several solutions allow limited interoperability with third-party clusters due to CNI or IP addressing requirements. Overall, Liqo appears to be the most complete and versatile option.

IV. Design of the Liqo Network Fabric

A. Overall architecture

The Liqo networking module is responsible for integrating with the CNI plugin deployed within each cluster and enabling IP-level communication among Pods distributed across the federated mesh. For every established peering relationship, the system provisions a dedicated gateway Pod within each participating cluster. These gateway Pods function as the termination points of a WireGuard tunnel, which is employed to securely transmit all inter-cluster traffic. WireGuard is utilized as the underlying VPN protocol due to its ability to ensure traffic confidentiality, making Liqo suitable for deploying over public network infrastructure. Additionally, WireGuard generally offers lower latency and higher throughput compared to traditional VPN solutions such as IPsec or OpenVPN, while relying on established and trusted encryption primitives [12], [13].

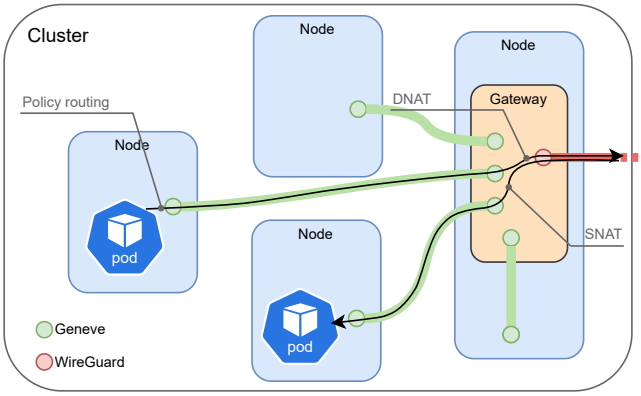

Fig. 1. Transmit and receive inter-cluster packet flow in Liqo.

Internally, the system establishes a mesh of Geneve tunnels interconnecting all Nodes within the cluster to each active Gateway, including the ones on which the Gateways themselves reside. This architecture enables the seamless routing of Pod traffic to and from the Gateway, independent of the source Node, by leveraging the native capabilities of the underlying CNI. While a non-encapsulated, flat networking model could theoretically offer improved performance, it is rendered impractical due to the need for robust interoperability with heterogeneous CNI implementations and configurations. Specifically, such a model would necessitate invasive modifications to the Linux routing table on every Node, potentially conflicting with the CNI's own configuration, particularly for those that are based on kernel-bypass technologies. By contrast, the encapsulated Geneve-based approach allows the Liqo network fabric to remain agnostic to CNI internals, while instead cooperating with the CNI and inheriting any traffic manipulation – such as strict encryption or traffic shaping – that it might be providing.

B. Address remapping and External CIDR

As previously discussed, Liqo provides full support for the federation of clusters configured with overlapping IP address ranges. This is achieved through the independent remapping of conflicting CIDR blocks on a per-cluster basis, managed by the dedicated *IPAM* component. Furthermore, Liqo introduces the concept of an "*External CIDR*", a reserved address range utilized for exposing external endpoints within

the federated mesh. These endpoints can include Pods residing in remote clusters or resources accessible via the Internet. The assignment and management of addresses within the External CIDR are handled via the `IP` Custom Resource Definition (CRD), which maps each endpoint to a specific external IP address, thereby enabling access from adjacent clusters. This mechanism is particularly useful in multi-peering topologies, where it facilitates inter-cluster communication – for example, allowing Pods in leaf clusters to communicate via a central, intermediary cluster. As such, Liqo automatically handles the creation of the appropriate `IP` CRDs for Pod or Service offloading use-cases performed through the Liqo control plane.

C. Pod-to-Pod inter-cluster traffic flow

Figure 1 illustrates the complete end-to-end flow of network packets during a typical cross-cluster Pod-to-Pod communication scenario. When a local Pod initiates communication with a remote Pod residing in a peered cluster, the outbound traffic is initially processed by the standard CNI mechanisms, which forwards it to the Node's networking stack. At this point, Liqo-specific policy routing rules intercept the traffic and redirect it through the Node's Geneve virtual interface. The packets are then encapsulated and forwarded by the underlying CNI to the local Liqo Gateway, which serves as the egress point towards the remote cluster. Here, Liqo performs any necessary destination NATting in accordance with the remapping settings being used; the traffic is subsequently encapsulated within the WireGuard VPN tunnel for secure transmission. Upon arrival at the remote cluster's Gateway, the packets are decrypted and decapsulated. A combination of routing and Netfilter rules then directs the traffic through the appropriate Geneve interface towards the destination Node. Finally, the CNI stack at the remote end delivers the traffic to the target Pod, completing the end-to-end communication path.

D. Service reflection

Liqo leverages the *Virtual Kubelet* [14] project to implement its control plane architecture, enabling the offloading of Pods and Services across cluster boundaries. Upon establishing a peering relationship, Liqo deploys a new Virtual Kubelet instance within the consumer cluster, which abstracts the entire provider cluster as a single, virtual Node. Kubernetes Pods created within specifically annotated namespaces can then be transparently offloaded to this virtual Node, effectively executing in the remote cluster. To support offloaded workloads, Liqo automatically performs *resource reflection*, synchronizing essential Kubernetes resources – such as Services, ConfigMaps, and Secrets – into the provider cluster. Notably, the Service reflection mechanism enables the extension of Kubernetes Services across federated clusters by replicating their specifications within each cluster's control plane. Corresponding EndpointSlices are also reflected, with address remapping based on the mesh configuration to ensure correct routing. This approach allows users to automatically expose Services to peered clusters while also enabling the creation of heterogeneous, cross-cluster Service backends for purposes such as load balancing and fault tolerance.

V. Current Limitations and Future Developments

As described so far, the Liqo network fabric enables seamless communication between Pods residing across a federated continuum of computing clusters, while adhering to native Kubernetes paradigms. Nevertheless, some notable limitations remain, which currently represent the most active areas of development within the project. The following subsections outline two such challenges, along with the ongoing efforts and planned enhancements aimed at addressing them.

A. Enforcing cluster boundaries via explicit Gateway filtering

Currently, Liqo is configured to accept all traffic originating from peered clusters, hence enabling unrestricted communication between any two Pods within the federated mesh – even when those Pods have not been explicitly offloaded via the Liqo control plane. This work-in-progress extension aims to enhance traffic control by allowing cluster operators to explicitly define and restrict inter-cluster communication. In fact, delegating this responsibility to native Kubernetes *Network Policies* is insufficient for several key reasons:

- **CNI independence.** Enforcement of Network Policies is not mandated by the CNI specification. Relying on the underlying CNI for policy enforcement would therefore conflict with Liqo's strive for CNI independence.
- **Single- vs multi-cluster scope.** Most CNIs are architected to operate within the context of a single cluster. As such, even CNIs that support Network Policies may not correctly handle rules targeting external or remote-cluster IP addresses.
- **Encapsulation constraints.** Liqo encapsulates inter-cluster traffic using Geneve tunnels at the source Node. This hides the original packet endpoints from the CNI, which can thus only observe traffic directed to the local Liqo Gateway, preventing accurate policy enforcement.

To address these limitations, the proposed solution integrates traffic filtering directly within the Liqo network fabric. This is achieved by extending the existing `FirewallConfiguration` custom resource – already used for specifying NAT and routing rules – to also support filtering actions. The associated controller, which operates within the Liqo Gateway Pod, translates these specifications into *nftables* rules installed in the Gateway's network namespace. As all cross-cluster traffic traverses both the local and remote Gateways, each cluster in a peering relationship is granted the ability to selectively permit or deny traffic destined for specific federated endpoints, ensuring consistent and CNI-agnostic enforcement.

B. Routing optimizations for inter-leaf traffic

In a multi-cluster deployment comprising one consumer ("*central*") and two or more provider ("*leaf*") clusters, Liqo enables leaf-to-leaf communication by forwarding their traffic through the consumer via automatic or manual `IP` resources

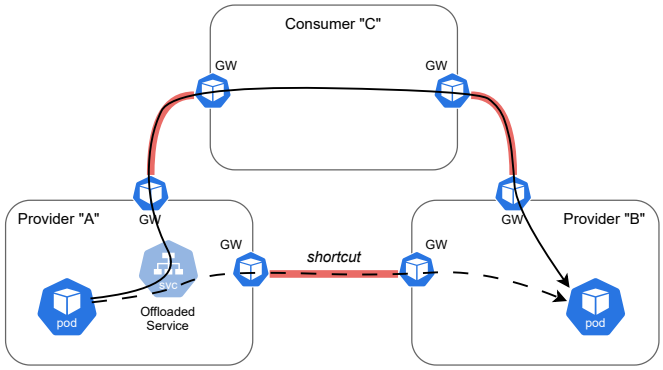

Fig. 2. Overview of the work-in-progress routing optimization feature for leaf-to-leaf traffic.

and external CIDR remapping, as described in Section IV. However, even in case a dedicated direct shortcut is established between two leaves, the Liqo fabric is currently incapable of taking advantage of it, and will continue to route inter-leaf packets through the central cluster. Although this situation is likely not common for cross-organization federations, it may prove problematic for large, distributed cloud deployments managed by the same entity, as the inability of directly routing traffic between providers may cause increased latency, aggravated load on critical network paths, and general inefficiency of the mesh network.

Liqo is thus exploring the possibility of remedying this shortcoming in a native and coherent way. Specifically, the proposed extension is twofold; first, it enhances the `liqoctl` helper with a new subcommand designed to facilitate and streamline the consumer-driven establishment of provider-to-provider shortcuts, hence mimicking the automation of the peering process implemented in Liqo. This is mediated by the new ad-hoc `ForeignClusterConnection` Custom Resource and related controller, which performs shortcut setup by provisioning dedicated Gateway pairs and CIDR remapping rules, similarly to how Liqo already handles the opening of new peering connections. The second contribution necessitated by the implementation of this feature is related to automating the use of the available shortcuts when using Liqo-offloaded Services. This requires enhancing the Service and Endpoint reflection logic with network shortcut awareness, and is a presently ongoing effort. Figure 2 depicts a leaf-to-leaf communication through an offloaded Service, together with the current Liqo behavior of transporting the traffic through the central consumer cluster (*solid line*), as well as the work-in-progress enhancement that takes advantage of the direct shortcut (*dashed line*).

VI. Conclusions

In this paper, we addressed the challenge of interconnecting distributed Kubernetes clusters within computing federations. We began by outlining the fundamental requirements that any networking solution must satisfy to support effective inter-cluster communication. Then, we provided a comprehensive overview of existing projects aimed at this functionality. Among them, Liqo was identified as the solution that most aligned with the identified requirements; as such, we presented a detailed technical analysis of its network fabric explaining the design behind its inter-cluster Pod traffic routing, CIDR remapping and resource reflection features, each contributing to its peering architecture. Finally, we discussed two active areas of ongoing development within the project: enhancing inter-cluster network security and policy enforcement, and optimizing routing through the introduction of leaf-to-leaf shortcuts.

References

[1] Cloud Native Computing Foundation and Linux Foundation Research, "Cloud native 2024: Approaching a decade of code, cloud, and change," https://www.cncf.io/reports/cncf-annual-survey-2024/, Cloud Native Computing Foundation, Annual Survey Report CNCF Annual Survey 2024, April 2025, survey conducted with 750 respondents during fall 2024.

[2] The Submariner Contributors. (2025) Submariner – multi-cluster networking for kubernetes. [Online]. Available: https://submariner.io/

[3] Kubernetes SIG-Multicluster. (2025) Kep 1645: Multi-cluster services api. [Online]. Available: https://github.com/kubernetes/enhancements/tree/master/keps/sig-multicluster/1645-multi-cluster-services-api/

[4] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *NDSS*, 2017, pp. 1–12.

[5] The Skupper Contributors. (2025) Skupper: Multicloud communication for kubernetes. An open-source Virtual Application Network enabling secure Layer-7 communication across Kubernetes clusters. [Online]. Available: https://skupper.io/

[6] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, "Computing without borders: The way towards liquid computing," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2820–2838, 2023.

[7] The Liqo Contributors. (2025) Liqo: Enable dynamic and seamless kubernetes multi-cluster topologies. An open-source solution for Kubernetes cluster peering, workload offloading, and multi-cluster networking and storage fabric. [Online]. Available: https://liqo.io/

[8] Isovalent. (2025) Cilium cluster mesh: Multi-cluster networking. Seamless multi-cluster pod and service connectivity, fault tolerance, global service discovery, and policy enforcement across Kubernetes clusters. [Online]. Available: https://cilium.io/use-cases/cluster-mesh/

[9] Tigera. (2025) Calico cluster mesh: Multi-cluster networking. Federated services, security compliance, observability, and cross-cluster connectivity. [Online]. Available: https://www.tigera.io/tigera-products/cluster-mesh/

[10] The Istio Authors. (2025) Istio multicluster. [Online]. Available: https://istio.io/latest/docs/setup/install/multicluster/

[11] The Linkerd authors. (2025) Linkerd multicluster. [Online]. Available: https://linkerd.io/2-edge/tasks/installing-multicluster/

[12] A. V. Ostroukh, C. B. Pronin, A. A. Podberezkin, J. V. Podberezkina, and A. M. Volkov, "Enhancing corporate network security and performance: A comprehensive evaluation of wireguard as a next-generation vpn solution," in *2024 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, 2024, pp. 1–5.

[13] S. Mackey, I. Mihov, A. Nosenko, F. Vega, and Y. Cheng, "A performance comparison of wireguard and openvpn," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 162–164. [Online]. Available: https://doi.org/10.1145/3374664.3379532

[14] The Virtual Kubelet authors. (2025) Virtual kubelet: Kubernetes node abstraction for serverless and multi-cluster providers. [Online]. Available: https://virtual-kubelet.io/