POLITECNICO DI TORINO
Repository ISTITUZIONALE

Intent-driven network isolation for the cloud computing continuum

Original
Intent-driven network isolation for the cloud computing continuum / Pizzato, F., Bringhenti, D., Sisto, R., Valenza, F.. - In: JOURNAL OF NETWORK AND SYSTEMS MANAGEMENT. - ISSN 1064-7570. - ELETTRONICO. - 34:1(2026), pp. 1-39. [10.1007/s10922-025-09986-1]

Availability:
This version is available at: 11583/3003532 since: 2025-10-30T08:56:20Z

Publisher:
Springer

Published
DOI:10.1007/s10922-025-09986-1

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

19 November 2026

Journal of Network and Systems Management (2026) 34:6
https://doi.org/10.1007/s10922-025-09986-1

Intent-Driven Network Isolation for the Cloud Computing Continuum

Francesco Pizzato[1] · Daniele Bringhenti[1] · Riccardo Sisto[1] · Fulvio Valenza[1]

Received: 28 March 2025 / Revised: 31 July 2025 / Accepted: 30 September 2025
© The Author(s) 2025

Abstract
The computing continuum is a revolutionary cloud paradigm that integrates edge, fog, and cloud layers into a cohesive distributed system of interconnected devices, enabling seamless resource sharing across heterogeneous environments and administrative domains. Its interwoven nature introduces novel challenges, including enforcing proper network isolation between workloads by managing all possible communications. Existing solutions are inadequate as they fail to address the dynamicity and heterogeneity of the computing continuum, exposing users to security risks like cross-tenant interference or side-channel attacks. To address these security challenges, this paper proposes a security solution to automate the configuration of network isolation across the computing continuum. The solution facilitates the enforcement of advanced security patterns, such as zero trust and least privilege, across the several cloud layers involved in the continuum. It employs an intent-based approach, enabling users to specify security requirements in an intuitive, high-level language. The process relies on two core phases: smart verification and harmonization, followed by translation. Their design aims to ensure consistency in the defined intents and adaptability in addressing the evolving nature of the continuum, by simplifying the configuration of advanced security patterns and providing tenants with fine-grained control over network isolation. The approach was implemented in Kubernetes, demonstrating its effectiveness in automating the enforcement of user-defined intents via Kubernetes Network Policies, a common mechanism for network isolation in Kubernetes. The developed implementation was validated both qualitatively in a comprehensive use case, confirming its effectiveness for security management, and quantitatively to assess the performance of the different phases of the process.

Keywords Cloud security · Kubernetes · Security automation

Extended author information available on the last page of the article

Published online: 17 October 2025

1 Introduction

Cloud computing has emerged as the principal paradigm for deploying and scaling large applications, simplifying the management of widely distributed and resilient services. Within this broad field, the computing continuum has recently gained significant traction as it permits bridging the gap between edge, fog, and cloud layers. Consolidating these layers into a cohesive, distributed architecture of interconnected devices facilitates the dynamic sharing of resources, such as network, data, or computation, enhancing flexibility and unlocking new optimization strategies [1–3]. Nonetheless, the computing continuum also introduces different security concerns.

A challenge yet to be addressed is guaranteeing isolation for the workloads deployed within the continuum, which could be declined into several problems such as access control [4], sandboxing [5], and network isolation. Specifically, this paper focuses on the network isolation problem related to managing and securing all communications within the use cases of the continuum. In this innovative environment, managing the network communications happening among the workloads (i.e., Kubernetes Pods) deployed in the continuum is of utmost importance to ensure proper isolation between tenants. For instance, if a user offers part of his cluster to others, it should be possible for him to manage all communications between hosted and local workloads to avoid undesired interference from hosted tenants. On the other hand, the user deploying a workload within the computing continuum may also desire control over all kinds of associated communications, independently of where source and destination are physically orchestrated. To properly address this problem, several sub-challenges need to be simultaneously addressed.

A primary concern is multi-tenancy. As the continuum resources are dynamically shared among users, clusters are exposed to risks such as cross-tenant interference and side-channel attacks. Moreover, recent reports [6] indicate that over 90% of surveyed Kubernetes clusters suffer from inadequate network security configuration, thereby increasing these risks. Furthermore, existing approaches rely on static and manual configurations, making them ill-suited for the dynamic and multi-tenant nature of the continuum, where users can fluidly expand their cluster over multiple distributed machines.

Secondly, the heterogeneity of the underlying infrastructure represents an additional complexity factor for the computing continuum. Variations in hardware and software components lead to significant differences in low-level network security configurations. For instance, the behavior of Kubernetes Network Policies depends upon the Container Network Interface (CNI) in use, with some of them, such as Flannel, lacking native policy enforcement and consequently silently ignoring them [7, 8]. This inconsistency complicates network security enforcement in the cloud and increases the risk of unintentional exposure of the system.

Finally, the management of network isolation in the continuum is made more complex due to the presence of multiple simultaneous objectives. The envisioned dynamic exchange of resources, i.e., users may borrow resources offered by others on remote clusters, implies different and potentially conflicting security objectives that must be balanced. In particular, each hosting cluster must be protected against potential harm caused by guest applications, which may even belong to tenants from

different administrative domains. In turn, guest applications are required to be safeguarded from potential stealing of data or code or undesired interference from the host. Instead, application owners must be empowered to define precise interactions, specifying what communications should be permitted or prohibited, while respecting the constraints of the hosting environments. As a consequence, any effective solutions must effectively balance these competing requirements while ensuring robust isolation across cloud layers.

In order to address these challenges, this paper proposes an automatic approach for the enforcement of network isolation across the computing continuum, addressing all the communications and use cases that may arise in such an environment. The proposed solution leverages an intent-based approach, allowing users to specify security requirements through a high-level user-friendly intent language. This language is capable of expressing all the aforementioned conflicting objectives. The solution harmonizes these intents, resolving possible conflicts between tenants or administrative domains and generating sets of intents that are instead conflict-free. The harmonization algorithm is designed to harmonize all possible discordances between the intents defined by multiple users, each with possibly different roles, which share the same physical cluster within the computing continuum. Ultimately, the harmonized intents are translated into actionable network security primitives, such as Kubernetes Network Policies, guaranteeing robust isolation across the continuum. The approach's effectiveness is demonstrated through its implementation and validation, showcasing its ability to automate intent translation and ensure consistent security configuration in dynamic, distributed environments. On the one hand, a qualitative validation based on a comprehensive use case illustrates the operational benefits of the proposed approach, simplifying the enforcement of advanced security patterns while maintaining strong tenant isolation. On the other hand, performance tests were carried out with the objective of providing a quantitative evaluation of the three algorithmic phases characterizing the proposed solution.

This paper is an extension of a preliminary conference paper published in the proceedings of the 2024 IEEE International Conference on Network Softwarization (NetSoft 2024) [9]. That version provided only a high-level overview of the approach and a basic description of the prototype implementation. Instead, this article introduces several key novelties:

- Enhanced definition of the proposed intent language;
- Refinement of the approach with the addition of a novel verification phase;
- Detailed formal presentation of the intent harmonization algorithm;
- An improved validation, assessing the effectiveness of the solution.

The remainder of this paper is structured as follows. Section 2 reviews related work. Sect. 3 presents the problem addressed by the proposed solution and its objective, and it also discusses the overall approach. Sect. 4 discusses the formalization of the intent language. Section 5 details the workflow, focusing on the core phases of harmonization and translation. Sect. 6 discusses the termination and correctness of the designed algorithms. Sect. 7 discusses the obtained results and the validation conducted on

the implementation. Finally, Sect. 8 concludes the paper and outlines future research directions.

2 Related Work

Network security automation has been explored in recent years to make use of the improved dynamism of virtual networks. In particular, a recent survey [10] analyzed the state of the art about the automation of network security configuration. Among the various studies conducted in this field, the most effective ones [11–15] successfully integrate configuration automation with policy-based management. This innovative approach brings different benefits, such as reducing the risk of misconfigurations and introducing the usage of high-level intents. Specifically, [11] proposes a new workflow for security configuration in virtual networks, introducing "projections" as abstractions of the defined network security policies to defer the selection of Virtual Network Functions (VNFs) with the goal of optimizing their cost and number. Although innovative for optimizing and managing VNFs, this work does not address the granular isolation of individual workloads (such as containers or microservices), focusing instead on the endpoint granularity of virtualized network environments. Moreover, it does not address the specific challenges of the computing continuum, such as the expression of application-centric network isolation policies or the simultaneous existence of conflicting objectives. In [12], INSpIRE is introduced, an intent-based networking solution that refines high-level intents into the configuration of VNF service chains, using clustering algorithms for VNF selection based on security requirements extracted from user intents. Even if it has a wider scope, since it supports both virtualized networks and heterogeneous environments composed of both softwarized components and physical middleboxes, it still focuses on network endpoints rather than the isolation of workloads and the definition of application-specific policies (e.g., all workloads belonging to the frontend service). Moreover, this work focuses on service composition and function selection rather than the management of the dynamic network isolation required in computing continuum environments. Instead, [13, 14] presents a methodology for automating the allocation and configuration of packet filters in virtual network topologies. Their approach is based on solving a Maximal Satisfiability Modulo Theories problem, ensuring automation, formal correctness, and optimality in terms of resource utilization. Specifically, [13] proposes a framework for automatically configuring virtual firewalls in Kubernetes. Although it applies to Kubernetes-based environments, the approach focuses on network-level firewalls, not delving into workload-specific isolation requirements. Moreover, it does not consider other fundamental challenges of the continuum, such as the presence of conflicting goals or the necessity of managing inter-cluster communications, being limited to scenarios with a single Kubernetes cluster. Similarly, the work presented in [14] represents a more mature and refined approach with respect to the previous one but it retains the same limitations. In particular, its focus remains on firewalls and does not extend to workload or application-level isolation requirements or the specifics of continuum computing. Finally, [15] presents an architecture for automatic provisioning of secure intent-based services through in-flight encryption

in SDN multilayer networks, i.e., IP, Ethernet, optical. The main contribution is the adoption of an intent-based networking approach for the automatic orchestration of the cryptographic layer based on application, costs, and security requirements. This work applies mainly to the SDN network context, and it aims to ensure confidentiality and integrity through encryption, rather than network isolation among the workloads. Overall, even if their contributions to the field of network security automation are highly valuable, none of these work is applicable to the problem of network-level workload isolation in the computing continuum, either because they focus on different security aspects, e.g., confidentiality, or because they are not designed to account for the specifics of such environments, e.g., multiple and concurrent goals, inter-cluster communications, workload-centric networking.

Security automation within the cloud thus represents a natural follow-up to what has already been investigated in that field. In this context, there have been different studies trying to automate security-related tasks. Several solutions [16–18] have been proposed for the automatic verification of user-defined security intents over formal models generated starting from configuration files of a cloud system. For instance, [16] employs description logics to model AWS CloudFormation configurations, i.e., the proprietary infrastructure-as-code solution of Amazon Web Services (AWS), in order to verify compliance and detect complex errors before workload deployment. Similarly, [17] introduces ZELKOVA, an analysis tool that translates AWS access control policies into Satisfiability Module Theories formulas to verify reachability and resource access properties. The goal is to aid administrators in analyzing the deployed configuration and detecting unintended misconfigurations. Additionally, [18] presents Tiros, which also leverages Satisfiability Module Theories formulas and automated theorem provers, but to model networking-related configurations within AWS cloud environments. The objective is to enable network reachability analysis on deployed applications in order to provide formal security guarantees about their adherence to some defined security invariance. All these tools, regardless of their specific focus on cloud environments, share the same limitation. Specifically, their goals are related to the analysis and verification of existing configurations (e.g., access control, IaC), not the automatic generation or dynamic application of network isolation policies across a heterogeneous computing continuum.

Other studies [19, 20] focus on the extraction of an enriched model from Kubernetes configuration files, so as to use it to solve different automated reasoning problems, such as attack graph generation and threat analysis. Specifically, [19] proposes a methodology for analyzing Kubernetes Helm Charts by generating a topological graph enriched with security features, which is used to assess risks using the MITRE ATT&CK framework. This work focuses on analyzing and identifying vulnerabilities prior to deployment, but does not contribute to automating the generation or enforcement of network isolation policies. Instead, [20] employs an enriched knowledge graph to assess vulnerabilities in cloud configurations and suggest more secure alternatives, supporting what-if analysis. Also in this case, although it does consider workflow-centric security aspects, the approach models the existing cloud configurations with the goal of providing an analysis and recommendation tool. It does not automate the synthesis or enforcement of security features. Despite their contribu-

tions, these solutions focus solely on security posture verification and compliance, lacking the capability to autonomously generate security configurations.

Only a few studies [21–23] try to automate the enforcement of low-level security configuration aspects, starting from the analysis of running systems. For example, [21] proposes Kub-Sec, an engine to generate AppArmor profiles for containers in Kubernetes clusters based on container runtime behavior. The system collects behavioral data from nodes, centrally transforms them into AppArmor policies, and applies them seamlessly without service disruption. This work is specific to the security of containers using AppArmor, which is a mandatory access control system based on resource access implemented in the Linux Kernel. Therefore, it is not applicable to the problem of workload-level network isolation, nor does it consider the requirements of the computing continuum. Similarly, [22] presents KGSecConfig, a knowledge graph-based approach for automating the security configuration of container orchestrators by systematically capturing, linking, and correlating the different heterogeneous sources of information in a unified structure. Some aspects addressed in this work, i.e., collection and aggregation of security configuration knowledge across heterogeneous sources, are highly relevant to the continuum, to which it shares similar heterogeneity properties. However, its focus is on general security compliance (e.g., presence of common misconfiguration, compliance with published best practices) rather than specifically on network isolation. Lastly, [23] proposes an automated tool for dynamically generating inter-service access control policies in microservices through static analysis of request patterns. Although highly relevant to the workload-isolation problem, its focus is more on access control rather than network isolation at a broader level. In general, all these solutions are limited in scope, as they do not support high-level security features such as Kubernetes Network Policies, are not designed to operate within the computing continuum paradigm, and, more importantly, lack user-defined intents. Indeed, the presence of multiple and conflicting objectives within the continuum imposes that users must have the power and flexibility to define the isolation intents.

As an overview, Table 1 summarizes this analysis of the related work, classifying the studies with respect to the network isolation challenges of the computing continuum and concisely highlighting their main limitations with respect to our proposal.

3 The Proposed Solution

This section illustrates the motivation and main features of the proposed solution for intent-driven network isolation in the computing continuum. First, it discusses the problem addressed by the proposed solution and its objective (Sect. 3.1). Then, it presents the approach followed by the designed methodology to reach the isolation goals (Sect. 3.2).

3.1 Problem Statement and Objective

In the computing continuum, resources can be dynamically acquired or released depending on users' evolving needs, generating a continuous exchange over the uni-

Table 1 Classification of related work with respect to network isolation challenges in the computing continuum

References	Security objective	Target environment	Automation type	User-defined intents	Workload scope	Multiple and concurrent goals
[11]	VNF selection	Virtualized networks	Configuration	✓	X	X
[12]	VNF selection	Virtualized and hybrid networks	Configuration	✓	X	X
[13]	Network isolation	Virtualized and cloud networks	Configuration	✓	X	X
[14]	Network isolation	Virtual networks	Configuration	✓	X	X
[15]	Network encryption	Multilayer SDN networks	Configuration	✓	X	X
[16]	Security assessment	Cloud (AWS)	Analysis	X	✓	X
[17]	Access control	Cloud (AWS)	Analysis	✓	✓	X
[18]	Network reachability	Cloud (AWS)	Analysis	✓	✓	X
[19]	Security assessment	Cloud	Analysis	X	✓	X
[20]	Security assessment	Cloud	Analysis	X	✓	X
[21]	Access control	Cloud (Kubernetes)	Configuration	X	✓	X
[22]	Access control	Container Orchestrators	Configuration	X	✓	X
[23]	Access control	Cloud (Kubernetes)	Configuration	X	✓	X

fied pool of available ones. In this process, users may assume different roles. Those who consume resources of a remote cluster act as consumers, while those offering resources for remote consumption act as providers. In this context, the term cluster refers to one or more machines that are logically grouped under a single control plane and run containerized applications. To denote if the cluster is administered by a provider or a consumer, in the article, the terms provider cluster and consumer cluster will be used for brevity. When a consumer acquires a resource, this resides physically within the provider's remote machine, but it is logically seen as integrated into the local cluster of the consumer. In this view, the meaning of local and remote clusters is derived from the subject of the phrase. If the subject is the consumer, the local cluster is the consumer cluster, and the remote cluster is the provider cluster. The vice versa is applied if the subject is the provider.

Through the dynamic creation and closure of these connections, the computing continuum model improves several real-world use cases that benefit from the more flexible and efficient infrastructure usage. For instance, if a participant operates an on-premise cloud infrastructure with commonly underutilized resources, they could temporarily lease parts of their infrastructure to other continuum participants, generating revenue while reducing infrastructure management expenses. Conversely, a participant experiencing temporary requirements, such as a spike in application traf-

fic or scheduled maintenance, could decide to temporarily lease remote resources and smoothly migrate their application without expanding their local physical infrastructure. Moreover, applications having strict requirements in terms of geographical location, either because of latency or due to legal obligations and compliances, could be seamlessly (re)orchestrated to different locations thanks to the creation of new connections.

However, an obstacle to the dynamic exchange of resources is their security. Specifically, it is of utmost importance to isolate workloads belonging to different users, permitting only communications that are approved by all involved actors. Considering the above use cases, the resource provider would keep ownership over her resources and define which communications are permitted, whereas the consumer would desire to isolate her workloads against undesired interferences from the provider or other co-located consumers, specifying exactly which workloads can communicate with others and how. Unfortunately, to the best of our knowledge, current technologies are not considering scenarios with multiple and concurrent goals, i.e., network isolation goals of both consumer and provider, supporting only single-cluster or single-user scenarios. Moreover, the heterogeneity and scale of the continuum impose new challenges not yet faced in current approaches. Having a vast pool of interconnected and different clusters implies a widely diverse landscape of network isolation solutions, which are not always intercompatible. But then, even if compatibility is assured, the isolation goals of consumers may contrast with those of the providers. For instance, a requested communication from a consumer workload to another hosted by the provider, e.g., a data source or an exposed service, might be denied by the provider or only partially approved. Similarly, a list of requested communications defined by the consumer might only be partially compliant with the ones requested by the provider, which could define some mandatory communications, e.g., for security or monitoring reasons, which must be included in the consumer's set. Noticeably, being the provider's intent defined for a general consumer, these conflicts are expected, and their automated discovery and reconciliation are necessary.

Therefore, the objective of this article is to safeguard workloads within the continuum. Specifically, it presents an approach for workload-level network isolation based on an intent language capable of expressing the network isolation goals of both consumers and providers. The approach is designed around three main phases, namely verification, harmonization, and translation. The first one is designed to verify the feasibility of consumer intents with the provider ones, checking if the consumer's request is compatible with the provider's offer. However, even if compatibility is verified, some discordances may still be present. For this reason, the proposed approach includes an automatic and intelligent harmonization phase to detect and solve any possible discordance. Finally, the heterogeneity of the underlying components is managed by the translation phase, which converts the network isolation intents into the specific implementation of the targeted cluster, abstracting away all complexities deriving from the computing continuum context.

3.2 The Approach

In view of these preliminary concepts, this paper proposes a methodology for automating the enforcement of network security isolation by managing the communications related to the workloads deployed within the computing continuum, whether owned by a consumer or a provider. The methodology is designed to be integrated with the resource acquisition process, meaning that network protection occurs after the selection of a provider and before resource consumption. This ensures that newly acquired resources are protected before any workload is deployed.

Specifically, the proposed approach works as follows, as also represented in Fig. 1. Initially, both parties taking part in a resource exchange define fine-grained security intents (Sect. 4), specifying which communications they want to allow or which authorizations they are willing to grant. This is done by both users involved in a resource exchange, i.e., the provider, left side of Fig. 1, and the consumer, right side of Fig. 1. The intents defined by the consumer are mainly related to the protection of its virtual cluster, i.e., logical cluster formed by the composition of resources physically present in the local and one or more remote clusters, while the intents defined by the provider are related to the protection of its resources and the communications allowed to cross the consumer's virtual boundary, i.e., logical and elastic perimeter of the virtual cluster that could shrink and expand based on workload demands. In Fig. 1, the virtual cluster is represented with a blue area that spans across both provider and consumer clusters, which are delimited by two white rectangles with a continuous black perimeter. Then, since different objectives may lead to conflicting intents, a smart harmonization process resolves all possible conflicts arising between actors participating in a given peering (Sect. 5.1). The result is a set of harmonized intents that undergoes a final process, the translation and enforcement phase. This allows the abstraction of the underlying hardware, converting the generic intents to the low-level configuration, which is properly applied (Sect. 5.2). These phases are managed by a component named secure border agent, represented as a yellow box within Fig. 1. This agent is deployed on all clusters taking part in the continuum and serves as the interface between users, who provides the intent sets, the orchestration platform, which is most commonly Kubernetes, and the technological stack enabling the continuum, which is referred to as computing continuum provider in the rest of the article and in Fig. 1 is modeled with a light-purple box.

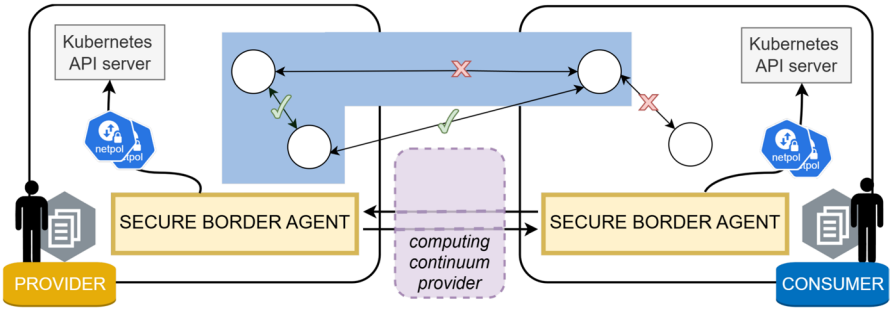


Fig. 1 high-level view of the protected border approach

A more detailed discussion on the intent language and the workflow is included in Sects. 4 and 5, respectively. The technological stack supporting the computing continuum must enable transparent resource consumption, i.e., ensuring that workloads can be scheduled either locally or remotely without any perceptible difference for the user. Additionally, since consumers and providers have simultaneous objectives, the continuum must include mechanisms for managing and sharing information across involved peers in order to allow the mutual sharing of intents representing their objectives. Section 3.2.1 further elaborates on these aspects.

Finally, the proposed approach assumes the existence of a minimal trust relationship between the consumer and the provider, which enables the exchange of security-related metadata and the negotiation of intents. This assumption reflects practical scenarios where resource sharing is governed by pre-established agreements, such as service-level agreements (SLAs) or contractual arrangements between administrative domains or tenants. Although the approach does not model the formal definition or enforcement of such agreements, their importance is undoubted for protecting provider configurations from potential misuse and preserving fairness in the intent negotiation process. The present work does not address these negotiation and trust problems, focusing instead on the workload isolation problem, which is a significant problem by itself to be addressed to improve the security of the computing continuum.

3.2.1 Computing Continuum Provider

The "computing continuum provider" architectural element is out of scope for this work, but as it plays a role in the proposed approach, its relevant characteristics are now briefly recalled.

The term computing continuum provider refers to the technological stack responsible for the creation of the different abstractions involved in the computing continuum. The most relevant abstractions are those concerning the unification of heterogeneous hardware and the transparent management and consumption of resources. In recent years, several initiatives have tried to propose different implementations of such component. Notably, several European projects [24–29] are currently working on different approaches, pushing the current technologies towards the next-generation computing continuum. Additionally, a broader European initiative[1] has been fostering collaboration among these projects, promoting a unified reference architecture for the European Cloud-Edge-IoT (ECEI) continuum . As reported in Fig. 2, this reference architecture consists of multiple modules that collectively provide all the abstractions needed for the continuum. Among these modules, resource management and orchestration are the most relevant for the present work. According to [30], the resource management module is responsible for resource discovery, orchestration, and managing co-allocation, i.e., the process of simultaneously allocating resources from different providers. Meanwhile, the orchestration module manages the deployment of services, applications, network resources, or even devices, across the cloud-edge continuum, abstracting the underlying infrastructure to facilitate seam-

[1] https://eucloudedgeiot.eu/.

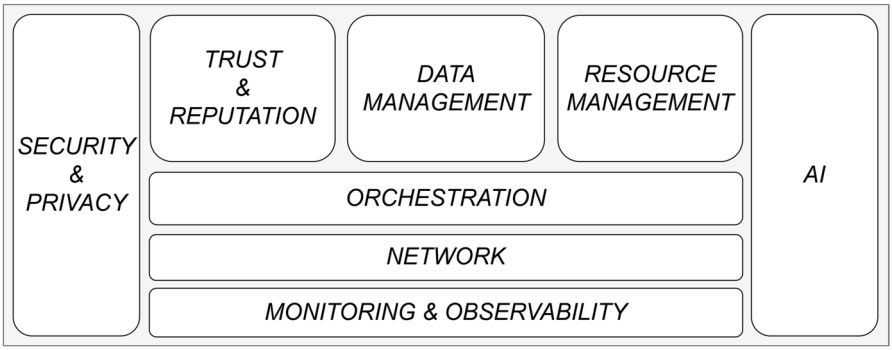


Fig. 2 ECEI reference architecture for the computing continuum

less consumption by users. It also manages the sharing of information across all the involved devices.

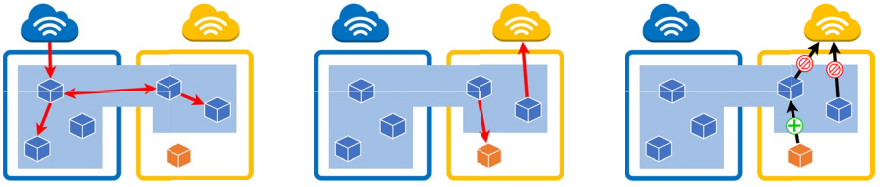
The approach in Sect. 3 only requires that the computing continuum provider exposes two key functionalities, i.e., the transparent consumption of resources, from their discovery up to their usage, and the ability to share knowledge between peers. Since these are part of the reference architecture, our proposed approach is general enough to be compliant with any computing continuum provider that adheres to the aforementioned reference document. For the purpose of this work, FLUIDOS [24] was adopted as a possible implementation of the computing continuum provider. Within FLUIDOS, resource management and data sharing are facilitated by Liqo [31]. This is an open-source project targeted at multi-cluster Kubernetes topologies and providing the technology for dynamic and seamless management of such environments, making it a good candidate to support the multi-federation model proposed within the computing continuum. Using Liqo, two heterogeneous clusters can establish a resource-sharing relationship. This relationship is called peering, and once a peering is established between two clusters, we will refer to them as peered clusters. Whenever a new peering is created, Liqo transparently handles all the low-level configurations in order to present a unified network and storage fabric to the user. Specifically, it permits multi-cluster connectivity regardless of the heterogeneity of the clusters and the interconnecting infrastructure, and it also supports transparent execution of stateful workloads independently of whether they are executed in a peered cluster or the other. Finally, once a peering is established, the process of scheduling a workload, i.e., a group of Kubernetes Pods executing an application, to the peered cluster is called offloading. To be aligned with Liqo's terminology, in the remainder of the paper, the process of connecting together two clusters will be called peering, and the operation of allocating a workflow in the remote node, connected through a peering, will be called offloading.

4 Intent Language for the Computing Continuum

A key element for the proposed solution is the intent language, which is introduced and formalized in this section before the one detailing the details of the workflow. A robust intent language for protecting the computing continuum must accommodate all the potential use cases within the novel scenarios of the continuum, while also enabling users to precisely define both permitted and prohibited network communications. Therefore, the purpose of the proposed intent language is to express all network security isolation requirements that are specific to the continuum, and allow users to define communication intents for pods across both physical and virtual clusters with the same fine granularity.

In order to achieve this objective, the proposed intent language involves four different types of intents, each tailored to achieve a specific objective, ensuring precise control over the communications happening in the continuum. The different types are linked to the roles of users involved in an offloading. When users assume the consumer role, their main security objective is to safeguard communications within their local cluster (Private intents) and to protect communications among resources offloaded to remote clusters (Request intents). Conversely, when users assume the provider role, their main security goal is to limit communications involving the hosted resources on the one hand, its own services or the external network (i.e., the Internet) on the other hand (Authorization intents). Moreover, the platform might require special networking privileges in order to work, and these must be also made explicit (Setup intents). These could be automatically derived from the infrastructure, therefore not being directly defined by either consumer or provider.

Figure 3 shows the three main types of intents, through some example communications. The intent types shown in the figure are only the ones under users' responsibility, which are Private and Request for the consumer, and Authorization for the provider. For what concerns intents of the the fourth type, i.e., Setup intents, they are correlated to the adopted platform and technological stack and could be derived from them. Consequently, being this type not direct responsibility of any user, this has not been represented. Besides, the consumer and provider roles may be interchangeably assumed by the users. Therefore, this subdivision of intent responsibilities simplifies the understanding of the approach, but it is not strict. Indeed, a provider has her own set of requests and private intent, but if she takes only the provider role, she does not define any intent for them, remaining empty. The opposite happens for the consumer



(a) Private intents (b) Request intents (c) Authorization intents

Fig. 3 User-defined types of intent

and the authorization intents. The possible types of intents are further detailed in the following:

- *Private intents*: they are related to communications within a virtual cluster, i.e., intra-virtual cluster, and involving both local and remote resources. These intents follow the principle that users should have full control over their resources, consequently they are not subject to the authorizations of the host(s). Some possible communications expressed with Private intents are represented in Fig. 3a. These intents are used to precisely define the intra-virtual cluster communications, allowing the implementation of a least privilege architecture.
- *Request intents*: they govern the inter-virtual cluster communications, i.e., those crossing the virtual boundary of an extended cluster. These may involve interaction with services in the hosting cluster or external networks. Within these intents, consumers can refine the isolation settings, tuning some parameters that will modify their will in terms of isolation with respect to the hosting cluster. For instance, these parameters could be used to accept or refuse to be monitored by the remote provider. Some examples are shown in Fig. 3b.
- *Authorization intents*: they represent authorization policies defined by each user to regulate inter-virtual cluster communication within a physical cluster, which is under they control (i.e., its under their administrative domain and have the possibility to modify its configuration). The communication policies defined through this type are applicable to any possible guest. Specifically, they target two communication types:

    1. *Denied Communication intents*: they express blocked or filtered connections for all hosts. For instance, the provider could choose to deny the connections to some blacklisted URLs, or to a subset of his resources.
    2. *Mandatory Communication intents*: they express whitelisted connections enforced for all guests. A possible usage is to inject into the configuration of each host a rule to permit the communications used by the provider to collect the logs of each hosted application.

Fig. 3cpresents some examples of communications that can be expressed with Authorization intents.

- *Setup intents*: ensure correct system operation by defining necessary platform-level networking configuration. For instance, these are used in our specific case to whitelist the VPN communications that are created between two peered clusters by Liqo.

All these intents can be expressed by the user with a format that provides a similar degree of expressiveness as the one achieved with selectors in Kubernetes Network Policies, so as to allow the specification of the information needed to select specific traffic. The envisioned structure is the following:

from SRC to DST, protocol [: port [− endPort]]

- *SRC* and *DST* can be either a pod or a group of pods with the same label and an associated namespace, or an address or a group of addresses defined through CIDR (at most one could be a CIDR address). For them, the wildcard symbol "*" can be used to target the whole (virtual) cluster if "*" is the value assigned to both pod and namespace, or to select all pods in a specific namespace if it is assigned only to the pod.
- *protocol* can be any transport protocol (TCP, UDP, SCTP, etc.) or the value "ALL" to represent all of them.
- *port* can be a port, or a range of ports if it includes also a value for *endPort*. This field is optional, and its omission has the same meaning as selecting the complete range of ports, i.e., 0-65535.

Finally, the Request and Private intents can be expressed only in whitelisting, so as to be compliant with the default behavior of Kubernetes Network Policies, which define the set of permitted communications and all the other ones are consequently blocked. Instead, the Authorization intents allow for more expressiveness and flexibility. The Denied Communication intents are expressed with a blacklisting approach, i.e., the user can define only denied communications and the remaining ones are allowed. Conversely, the Mandatory Communication intents are simply allowed communications, since they express the connections that must be imposed on all guests.

5 Workflow Formalization

As introduced in Sect. 3, the proposed approach is integrated into the resource acquisition process. During it, the consumer and the provider exchange the different sets of intents, as defined in Sect. 4. This is possible thanks to the abstractions provided by the computing continuum provider. Afterward, initial processing is executed by the cluster requesting the peering, i.e., consumer-side. Specifically, it verifies the compatibility of consumer intents with the authorization intents offered by all possible providers (Sect. 5.1.1). This phase can be performed multiple times, one for each received offer. Considering the results of this phase, one provider is selected, and the peering is initiated. Once this happens, a harmonization process (Sect. 5.1.2) is started within the secure border agent on the hosting cluster. As a result, the intents of the involved peers are harmonized and all possible conflicting objectives are solved. Finally, these harmonized intents are translated (Sect. 5.2) into proper configuration files for Kubernetes.

The workflow is graphically represented in Fig. 4, which illustrates a consumer cluster, on the left, performing the peering towards a provider cluster, on the right. As first step of the workflow, both users have to define the sets of intents under their respective competence, which are Request and Private intents for the consumer, while Authorization and Setup intents for the provider (1). As previously mentioned,

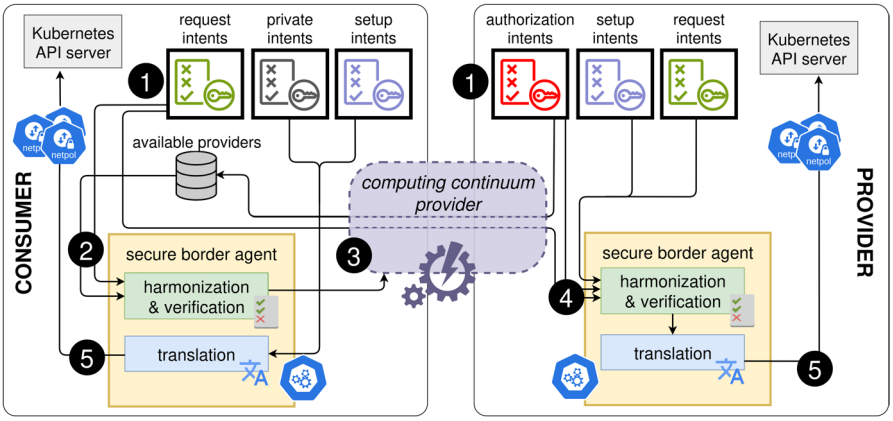


Fig. 4 workflow of the protected border approach

the provider also provides a set of request intents, whose content is determined by prior exchanges or possibly empty. As a hypothesis for the presented approach, the intent sets are supposed to be defined in an anomaly-free way. This is not restrictive because there are techniques that allow one to generate such a set from non-disjoint rules. Secondly, the secure border agent on the consumer cluster, after having collected the necessary information from all available providers, performs the verification phase (2). As a result, one provider will be selected by interfacing with the computing continuum provider (3). Note that this component is also invoked to share the consumer's intents with the provider. Afterward, the secure border agent on the provider is triggered and performs the harmonization phase (4). The output will be a set of conflict-free requirements, which is passed down to the translation module. This element handles their translation into Kubernetes Network Policies and their enforcement by contacting the API server of the corresponding cluster (5). This step happens in both clusters, as the consumer's Private intents, by definition, concern the intra-virtual cluster communications, and these could be either local or remote. As a consequence, some of them require to be translated and enforced in the local cluster, and others on the remote cluster.

In the following, the two modules inside the secure border agent will be explained in more detail.

5.1 Verification and Harmonization

The verification and harmonization module plays a crucial role in balancing the diverse and simultaneous objectives within the computing continuum, while ensuring the desired security level. The following sections will provide details about these processes. Remark that, for brevity, all symbols used in the next sections are introduced and defined in Table 2. Moreover, about the formalism of next sections, this work uses the "." notation to denote a specific tuple element (e.g., given a tuple $t = (a, b, c)$, $t.a$ identifies element a of tuple t).

Table 2 Policy specification grammar

Symbol/function/predicate/operator	Definition
$\mathbb{B} = \{true, false\}$	Boolean set
$\mathbb{S}$	Set of all finite-length strings
IP	Set of all valid IPv4 addresses
$\mathbb{P}$	Set of all Kubernetes Pods
$\mathcal{I}_{\mathcal{R}} = \{i_1, ..., i_m\}$	Set of Request intents
$\mathcal{F}_{\mathcal{R}} = \{x \mid x \in \mathbb{B}\}$	Set of parameters complementary to the Request intents
$\mathcal{I}_{\mathcal{P}} = \{i_1, ..., i_n\}$	Set of Private intents
$\mathcal{I}_{\mathcal{S}} = \{i_1, ..., i_p\}$	Set of Setup intents
$\mathcal{I}_{\mathcal{A}} = \mathcal{I}^d_{\mathcal{A}} \cup \mathcal{I}^m_{\mathcal{A}}$	Set of Authorization intents
$\mathcal{I}^d_{\mathcal{A}} = \{i_1, ..., i_q\}$	Set of Denied Communication intents
$\mathcal{I}^m_{\mathcal{A}} = \{i_1, ..., i_r\}$	Set of Mandatory Communications intents
$i = (C, a)$	Intent
$a \in \{allow, deny\}$	Action of an intent
$C = (proto, p, src, dst)$	Condition of an intent
$proto \subset \mathcal{V}$	Set of transport protocols
$\mathcal{V} = \mathcal{P}(\{STCP, UDP, TCP\}) \backslash \{\emptyset\} \cup \{ALL\}$	Set of all possible subsets for proto
$p \in \{(p_i, p_f) \mid p_i, p_f \in \{x \in \mathbb{N} \mid 0 \leq x \leq 65535\}, p_i \leq p_f\} \cup \{*\}$	Destination port
$src = (selector, \text{isRemoteCluster})$	Source selector
$dst = (selector, \text{isRemoteCluster})$	Destination selector
$\text{isRemoteCluster} \in \mathbb{B}$	Value indicating the cluster owning the resource
$selector \in \{S_{PodNs}, S_{CIDR}\}$	Resource selector
$S_{\text{CIDR}} = a_1.a_2.a_3.a_4/m$	Selector based on classless inter-domain routing value
$a_i \in \{x \in \mathbb{N} \mid 0 \leq x \leq 255\}$	IP octet
$m \in \{x \in \mathbb{N} \mid 0 \leq x \leq 32\}$	IP mask
$S_{\text{PodNs}} = (pod, ns)$	Selector based on pod and namespace values
$pod = \{(k_1, v_1), ..., (k_j, v_j)\}$	Set of key-value pairs for Kubernetes pods
$ns = \{(k_1, v_1), ..., (k_j, v_j)\}$	Set of key-value pairs for Kubernetes namespaces
$k_i \in \mathbb{S} \cup \{*\}, v_i \in \mathbb{S} \cup \{*\}$	Key and value
$\mathcal{BC} : S_{\text{CIDR}} \rightarrow IP$	Function mapping $\mathcal{S}_{\text{CIDR}}$ to its broadcast address
$\mathcal{NW} : S_{\text{CIDR}} \rightarrow IP$	Function mapping $\mathcal{S}_{\text{CIDR}}$ to its network address
$\phi : \mathcal{S}_{\text{PodNs}} \rightarrow \mathcal{P}(\mathbb{P})$	Function mapping $\mathcal{S}_{\text{PodNs}}$ to Kubernetes Pods

5.1.1 Verification

The verification phase has the purpose of ensuring the compatibility between a set of Request intents $\mathcal{I}_{\mathcal{R}}$ defined by a consumer, and a set of Authorization intents $\mathcal{I}_{\mathcal{A}}$ defined by a provider. This process will be repeated for each one of the available providers, as the goal is to provide results relevant to the consumer to assist her in the selection of a provider from the set of available ones. Multiple different strategies may be pursued to reach this objective, e.g., some strategies may allow the consumer to rank the results according to some criteria, selecting the provider whose Authori-

zation intents are most aligned with her Request intents. However, as the definition of all these strategies is not within the scope of this paper, the verification strategy defined in the current proposal simply performs a binary compatibility check. At the same time, the overall workflow has been designed to be compliant with the integration of other possible strategies, thus leaving the possibility of having more defined mechanisms for future work.

The verification algorithm takes as input each possible pair of intents (i_r, i_a) such that an element of the pair is a Request intent, whereas the other one is a Denied Communication intent, i.e., $(i_r \in \mathcal{I}_{\mathcal{R}}, i_a \in \mathcal{I}^d_{\mathcal{A}})$. For each input pair thus defined, it outputs a Boolean value, which is true if the intents of the evaluated pair are *compatible*, false otherwise. Two intents are compatible if there is no overlap between their conditions. Consequently, the verification algorithm can be considered as a verification of the existence of a possible overlap between the conditions of intents composing the input pairs, as shown in the following:

$$\forall i_r \in \mathcal{I}_{\mathcal{R}}, \forall i_a \in \mathcal{I}^d_{\mathcal{A}},\ i_r.C \cap i_a.C = \emptyset$$

The satisfaction of the above formula implies that the compared intent sets, i.e., $\mathcal{I}_{\mathcal{R}}, \mathcal{I}^d_{\mathcal{A}}$, are compatible. In this work, a set-theoretic interpretation is given to the intent's condition. In particular, given a condition $C = (proto, p, src, dst)$, this is interpreted as a multi-dimensional hypervolume described by four different dimensions, i.e., *proto*, *p*, *src*, and *dst*, and C is the set of elements included in this hypervolume. Therefore, the intersection $i_r.C \cap i_a.C$ is interpreted as the intersection of such hypervolumes in this interpretation. Table 3 presents a clarifying example of the verification process. In particular, Table 3 presents two sets of intents, namely a set of Authorization intents $\mathcal{I}^d_{\mathcal{A}}$ defined by a provider and a set of Request intents $\mathcal{I}_{\mathcal{R}}$ defined by a consumer. In particular, comparing the only Request intent i_r and $i_1 \in \mathcal{I}^d_{\mathcal{A}}$, it may seem that these intents are overlapping. However, even if several fields are clearly overlapping, i_1 and i_r are compatible because of the disjoint *proto* field. It is sufficient that one of the fields defining the intents is non-overlapping to have a true output for the verification. Instead, comparing i_r and $i_2 \in \mathcal{I}^d_{\mathcal{A}}$, if *proto* and *p* are ignored being evidently overlapping, the situation is the one of a partial overlap with respect to the *dst*. This means that the requested intent i_r is only partially authorized by the provider intents. Finally, the last case is the comparison between i_r and $i_3 \in \mathcal{I}^d_{\mathcal{A}}$. This situation presents a full overlap between the two intents. Specifically, the provider's intents are defined in such a way that *src* and *dst* are of wider-scope with respect to i_r, which results to be a subset of i_3.

Table 3 Example of intent sets to be verified

#	*src*	*dst*	*p*	*proto*
Request intents ($\mathcal{I}_{\mathcal{R}}$) of consumer				
1	$\mathcal{S}_{\text{PodNs}} = (\{(role, student)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/16$	(80, 80)	{TCP}
Denied communication intents ($\mathcal{I}^d_{\mathcal{A}}$) of provider				
1	$\mathcal{S}_{\text{PodNs}} = (\{(*, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/16$	$*$	{UDP}
2	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/18$	(80, 80)	{TCP}
3	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.0.0.0/8$	$*$	{ALL}

Additionally, the verification algorithm evaluates also any complementary parameter within the set of Request intents, as defined in Sect. 4. An example of a parameter is the Boolean flag $\mathrm{acceptMandatory} \in \mathcal{F}_{\mathcal{R}}$, which is true if the consumer is willing to accept certain connections imposed by the provider (e.g., monitoring connections for security or accounting purposes), false otherwise.

In checking the compatibility between the two intents composing each input pair, the most complex aspect of the verification algorithm is related to the fact that two intents are comparable only if they are defined using the same fields, i.e., if *C.src* and *C.dst* are expressed using the same *selector* types for both intents. Consequently, the verification algorithm employs a different processing logic depending on how they are specified, either through a CIDR-based *selector* or through a Kubernetes-based *selector*. Moreover, the comparison of two selectors of type $\mathcal{S}_{\mathrm{PodNs}}$ is non-trivial, requiring an additional processing step in order to make them comparable. As in Kubernetes a Pod might have multiple assigned labels, a simple string comparison of the list of (*k*, *v*) pairs composing the *pod* and *namespace* elements of the two *selector* may not be sufficient, and may lead to incorrect results. Furthermore, since the *selector* is expressed both in terms of pod labels, i.e., *pod*, and namespace labels, i.e., *ns*, comparing two selectors such as $\mathcal{S}^1_{\mathrm{PodNs}} = (pod, \emptyset)$ and $\mathcal{S}^2_{\mathrm{PodNs}} = (\emptyset, ns)$, would not be possible without this preliminary mapping. This operation is formally represented here with function $\phi(\mathcal{S}_{\mathrm{PodNs}}) = Pod$, which takes as input a $\mathcal{S}_{\mathrm{PodNs}}$ selector and produces as result a set of Kubernetes pods, i.e., $Pod \subset \mathbb{P}$. Specifically, this represents a mapping function that converts a resource selector, defined in terms of Kubernetes' Pod and Namespace labels, to the set of objects identified by those labels. In other words, given $S_{\mathrm{PodNs}} = (\{(k_1^{pod}, v_1^{pod})\}, \{(k_1^{ns}, v_1^{ns})\})$, the function $\phi(\mathcal{S}_{\mathrm{PodNs}})$ will return a set of Kubernetes Pods having a label with $key = k_1^{pod}$ and $value = v_1^{pod}$ and belonging to any namespace having a label with $key = k_1^{ns}$ and $value = v_1^{ns}$. This mapping is further complicated by the expressiveness of S_{PodNs} which could include the wildcard symbol in any position, with the extreme cases of selecting a complete (virtual) cluster, i.e., $S_{\mathrm{PodNs}} = (\{(*, *)\}, \{(*, *)\})$. The considerations presented here are valid also for the next part, where they will be applied within the formal presentation of the harmonization algorithm.

5.1.2 Harmonization

The harmonization phase is executed within the same module and has the purpose of harmonizing different sets of intents established by users engaged in a peering process, providing an intelligent resolution of the possible discordance between them. The general principle behind this algorithm is that the hosting cluster has the decision power: it chooses which Request intents, defined by the consumer, can or cannot be enforced while possibly forcing some new ones. Anyhow, the host can impose its authority only over the inter-virtual cluster communications, while the intra-virtual cluster communications, even if happening on the host's cluster, should be fully determined by the consumer who acquired the usage of those resources and has the right to configure them freely. These communications are expressed through a set of

Private intents, and, given this consideration, they can not produce discordances and are not involved in the harmonization process.

In the context of this phase, a discordance arises when an intent defined by one user is not authorized or coherent with intents defined by another user with whom peering is requested. These discordances can be classified into three main types related to three possible cases:

1. When a Request intent defined by the consumer is not (fully or partially) authorized by the Authorization intents of the provider;
2. When a Mandatory Communication intent defined by the provider is not satisfied (fully or partially) by the Request intents of the consumer;
3. When a Request intent defined by the consumer has been authorized but does not have a symmetrical Request intent in the provider.

This last case is needed to have coherence between the consumer and provider intent sets, since it is not enough that the consumer's Request intent is authorized but a similar provider's intent should be defined in order to fully allow the communication. These three discordance types are represented in Fig. 5 through an example. For the first case, a user performing the offloading is requesting that his offloaded entity A can contact a malicious website, but the Authorization intents defined by the hosting user deny all connections to the Internet for all offloaded pods, thus causing a discordance. Second, the provider defined a Mandatory Communication intent from his resource M, a monitoring endpoint, to all offloaded pods. This is not yet satisfied by the intents defined by the consumer, causing another discordance. Third, the consumer requests that the same offloaded entity A can contact entity B, which is part of the hosting cluster. However, the hosting user has not defined an intent allowing B to be contacted by A, thus resulting in another discordance.

The harmonization algorithm is divided into three main stages, one for each discordance type. These three stages, whose formalization is presented in the remainder of this section, must be executed sequentially, i.e., first, the discordances of type-1 must be harmonized, then the ones of type-2, and finally, the ones of type-3. This ordering has been selected in order to assure the correctness of the result. More precisely, the processing of type-3 discordances must be postponed to the last stage because one of its inputs is the final set of harmonized consumer's Request intents, which results from the resolution of type-1 and type-2 discordances. Moreover, harmonizing type-2 discordances before type-1 ones may cause some mandatory communications, added as a result of the harmonization of discordances of type-2, to be later removed because

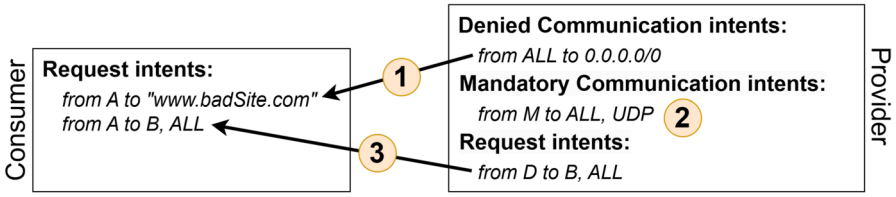


Fig. 5 Example of possible discordance between intents

they are in conflict with the provider's authorizations, produced instead a result of the harmonization of discordances of type-1. As a consequence, the only correct ordering of discordances for the execution of the harmonization algorithm on them is: type-1, type-2 and type-3.

Harmonization of discordances of type-1. Discordances of type-1 are managed through algorithm 1. Discordances of the first type are solved by subtracting the set of Denied Communication intents $\mathcal{I}_{\mathcal{A}}^{d}$ from the set of Request intents $\mathcal{I}_{\mathcal{R}}$. Indeed, the algorithm processes each tuple (i_r, i_a), with $i_r \in \mathcal{I}_{\mathcal{R}}$ and $i_a \in \mathcal{I}_{\mathcal{A}}^{d}$, and, if $i_r.C \cap i_a.C \neq \emptyset$ then the difference is computed as $\{C_1, ...C_k\} \leftarrow i_r.C \setminus i_a.C$, which produces a set of intents, one for each $C_j \in \{C_1, ..., C_k\}$. This procedure is then repeated for all tuples until all discordances are solved. This operation is done by the recursive function $Harmonize(i_r, \mathcal{I}_{\mathcal{A}}^{d})$. This procedure first evaluates the presence of overlap between the considered tuple (i_r, i_a), and, if present, the difference of their conditions is computed over all dimensions. Then, depending on the result of this operation, the function is recursively called on none or multiple subsets of $i_r.C$, which do not overlap with the condition of the currently considered i_a. Instead, if there is no overlap, the procedure proceeds to the other elements in $\mathcal{I}_{\mathcal{A}}^{d}$. If i_r does not cause overlap with any element in the set, it could be added to the set of harmonized Request intents $\mathcal{I}_{\mathcal{R}}^{\mathcal{H}}$. The recursive aspect is needed since an intent could produce a discordance with multiple authorization intents, and the procedure should be repeated until all of them are solved. To avoid redundant code within the algorithm, the function $harmonizeEndpoint(ep_1, ep_2)$ has been introduced to compute the difference along the components *C.src* and *C.dst*, which processing is identical.

Input: a set of Authorization intents $\mathcal{I}_{\mathcal{A}} = (\mathcal{I}^{d}_{\mathcal{A}}, \mathcal{I}^{m}_{\mathcal{A}})$, a set of Request intents $\mathcal{I}_{\mathcal{R}}$
Output: the harmonized set of request intents $\mathcal{I}^{\mathcal{H}}_{\mathcal{R}}$

$\mathcal{I}^{\mathcal{H}}_{\mathcal{R}} \leftarrow \emptyset$
for $i_r \in \mathcal{I}_{\mathcal{R}} = \{i_1, ..., i_m\}$ **do**
  $\mathcal{I}^{\mathcal{H}}_{\mathcal{R}} \leftarrow \mathcal{I}^{\mathcal{H}}_{\mathcal{R}} \cup$ HARMONIZE$(i_r, \mathcal{I}^{d}_{\mathcal{A}})$
end for
return $\mathcal{I}^{\mathcal{H}}_{\mathcal{R}}$

procedure HARMONIZE$(i_x, \mathcal{I}_y)$
  $tmpSet \leftarrow \emptyset$
  for $i_y \in \mathcal{I}_y = \{i_1, ..., i_q\}$ **do**
    $C_x \leftarrow i_x.C$; $C_y \leftarrow i_y.C$
    $dirty \leftarrow false$
    if $C_y.proto \cap C_x.proto \neq \emptyset$ **then**
      $protoSet \leftarrow C_x.proto \setminus C_y.proto$
    end if
    if $(C_x.p = * \vee C_y.p = *) \vee (C_x.p.p_i \leq C_y.p.p_f \wedge C_x.p.p_f \geq C_y.p.p_i)$ **then**
      $pSet \leftarrow C_x.p \setminus C_y.p$
    end if
    $src_x \leftarrow C_x.src$; $src_y \leftarrow C_y.src$
    $srcSet \leftarrow$HARMONIZEENDPOINT(src_x, src_y)
    $dst_x \leftarrow C_x.dst$; $dst_y \leftarrow C_y.dst$
    $dstSet \leftarrow$ HARMONIZEENDPOINT(dst_x, dst_y)
    if $(protoSet \neq C_x.proto) \wedge (pSet \neq C_x.p) \wedge (srcSet \neq C_x.src) \wedge (dstSet \neq C_x.dst)$ **then**
      $dirty \leftarrow true$ ▷ i_x overlaps with i_y, so should be removed from $\mathcal{I}^{\mathcal{H}}_{y}$
    else
      continue ▷ i_x does not overlap with current i_y, check next one
    end if
    if $protoSet \neq \emptyset$ **then**
      $i'_x.C.proto \leftarrow protoSet$
      $tmpSet \leftarrow tmpSet \cup$ HARMONIZE$(i'_x, \mathcal{I}_y)$
    end if
    if $pSet \neq \emptyset$ **then**
      for $p_i \in pSet$ **do**
        $i'_x.C.p \leftarrow p_i$
        $tmpSet \leftarrow tmpSet \cup$ HARMONIZE$(i'_x, \mathcal{I}_y)$
      end for
    end if
    if $srcSet \neq \emptyset$ **then**
      for $src_i \in srcSet$ **do**
        $i'_x.C.src \leftarrow src_i$
        $tmpSet \leftarrow tmpSet \cup$ HARMONIZE$(i'_x, \mathcal{I}_y)$
      end for
    end if
    if $dstSet \neq \emptyset$ **then**
      for $dst_i \in dstSet$ **do**
        $i'_x.C.dst \leftarrow dst_i$
        $tmpSet \leftarrow tmpSet \cup$ HARMONIZE$(i'_x, \mathcal{I}_y)$
      end for
    end if
  end for
  if $dirty$ **then**
    return $tmpSet$
  else
    return $tmpSet \leftarrow tmpSet \cup i_x$
  end if
end procedure

procedure HARMONIZEENDPOINT(ep_1, ep_2)
  $set \leftarrow \emptyset$
  if $ep_1.isRemoteCluster \neq ep_2.isRemoteCluster$ **then**
    if $ep_1.selector == ep_2.selector == S_{CIDR}$ **then**
      if $(\mathcal{NW}(ep_1.S_{CIDR}) \leq \mathcal{BC}(ep_2.S_{CIDR})) \wedge (\mathcal{BC}(ep_1.S_{CIDR}) \geq \mathcal{NW}(ep_2.S_{CIDR}))$ **then**
        $set \leftarrow ep_1.S_{CIDR} \setminus ep_2.S_{CIDR}$
      end if
    else if $ep_1.selector == ep_2.selector == S_{PodNs}$ **then**
      if $ep_1.S_{PodNs} \cap ep_2.S_{PodNs} \neq \emptyset$ **then**
        $set \leftarrow ep_1.S_{PodNs} \setminus ep_2.S_{PodNs}$
      end if
    end if
  end if
  return set
end procedure

Algorithm 1 Algorithm for harmonization phase (discordances of type 1)

In algorithm 1, one key operation is the difference between two intents' conditions. Since every condition consists of multiple fields, each one having a different value, this operation is not trivial since it has to be adapted to each field. Now, it is detailed how the difference is expressed for all the possible fields of C. Moreover, the formulas presented here are the same used in Algorithm 1. Introducing them here allows us to simplify its formulation. Let us consider two intents $i_1 = ((proto_1, p_1, src_1, dst_1), a_1)$ and $i_2 = ((proto_2, p_2, src_2, dst_2), a_2)$. The difference i_1 minus i_2 with respect to the transport protocol, i.e., *proto*, is defined as:

$$proto_1 \backslash proto_2 = \begin{cases} \emptyset & \text{if } proto_1 \subseteq proto_2 \text{ or } proto_2 = \text{ALL} \\ V \backslash proto_2 & \text{if } proto_1 = \text{ALL} \\ proto_1 \backslash proto_2 & \text{otherwise} \end{cases}$$

Instead, the difference of i_1 minus i_2 considering the destination port, i.e., p, is:

$$p_1 \backslash p_2 = \begin{cases} \emptyset & \text{if } p_1 \subseteq p_2 \text{ or } p_2 = * \\ \{x \in \mathbb{N} | 0 \leq x \leq 65535\} \backslash p_2 & \text{if } p_1 = * \\ (p_{i1}, p_{i2} - 1) \cup (p_{f2} + 1, p_{f1}) & \text{if } p_2 \subset p_1 \\ (p_{i1}, p_{i2} - 1) & \text{if } p_1 \cap p_2 \neq \emptyset \text{ and } p_{i2} < p_{f1} \leq p_{f2} \\ (p_{f2} + 1, p_{f1}) & \text{if } p_1 \cap p_2 \neq \emptyset \text{ and } p_{i1} < p_{f2} \leq p_{f1} \\ p_1 & \text{if } p_1 \cap p_2 = \emptyset \end{cases}$$

Finally, the difference of i_1 and i_2 with respect to *src* and *dst* is defined in the same way for both. This operation has a different formulation depending on the selector type, i.e., S_{CIDR} or S_{PodNs}. Starting with S_{CIDR}:

$$S^1_{\text{CIDR}} \backslash S^2_{\text{CIDR}} = \begin{cases} \emptyset & \text{if } S^1_{\text{CIDR}} \subseteq S^2_{\text{CIDR}} \text{ or } S^2_{\text{CIDR}} = \mathbb{I} \\ \mathbb{I} \backslash S^2_{\text{CIDR}} & \text{if } S^1_{\text{CIDR}} = \mathbb{I} \\ S^1_{\text{CIDR}} & \text{if } S^1_{\text{CIDR}} \cap S^2_{\text{CIDR}} = \emptyset \\ \bigcup S_i & \text{where } S_i \text{ are disjoint CIDR blocks covering } S^1_{\text{CIDR}} \backslash S^2_{\text{CIDR}} \end{cases}$$

In this formulation, $\mathbb{I}$ represents the space of all possible IP addresses. In the last case, where $S^1_{\text{CIDR}} \cap S^2_{\text{CIDR}} \neq \emptyset$ and $S^1_{\text{CIDR}} \not\subseteq S^2_{\text{CIDR}}$, the result is made up of different CIDR blocks resulting from the subnetting operation applied recursively to S^1_{CIDR}. Instead, in the case of S_{PodNs}, the difference is defined as:

$$S^1_{\text{PodNs}} \backslash S^2_{\text{PodNs}} = \phi^{-1}\big(\phi(S^1_{\text{PodNs}}) \backslash \phi(S^2_{\text{PodNs}})\big)$$

where the symbol ϕ^{-1} represents the inverse of function ϕ, so the mapping from a set of Kubernetes pods to a selector of type $\mathcal{S}_{\text{PodNs}}$. Note that, in the case where i_1 and i_2 are defined using different selector types, they are not comparable and thus they are not overlapping. In this situation, the difference is $i_1 \setminus i_2 = i_1$. Moreover, for the dimension p, as with *src* and *dst*, the difference between i_1 and i_2 may produce from zero up to n subsets of i_1 depending on the situation. Instead, for *proto*, the difference returns only zero or one subset.

Harmonization of discordances of type-2. Discordances of type-2 are processed in a partially similar way as the ones of type-1. For what concerns this second class of discordances, the purpose is to add, in an anomaly-free way, all the Mandatory Communication intents to the result of the previous harmonization stage, so the set of (partially) harmonized Request intents. This is accomplished by Algorithm 2 by, first, evaluating which intents in $\mathcal{I}_{\mathcal{A}}^{m}$ are not yet present in $\mathcal{I}_{\mathcal{R}}^{\mathcal{H}}$, i.e., $\mathcal{I}_{\mathcal{A}}^{m} \setminus \mathcal{I}_{\mathcal{R}}^{\mathcal{H}}$, and, second, add them to the final set of harmonized Request intents. In other words, the final set of harmonized Request intents is the composition of the Request intents with the Mandatory Communication intents, i.e., $\mathcal{I}_{\mathcal{R}} \leftarrow \mathcal{I}_{\mathcal{R}} \cup (\mathcal{I}_{\mathcal{A}}^{m} \setminus \mathcal{I}_{\mathcal{R}})$. The difference operation is needed in order to avoid redundancy in the resulting set of intents. Notice that the $Harmonize()$ function used in Algorithm 2 is the same as Algorithm 1.

Harmonization of discordances of type-3. Discordances of type-3 are handled by Algorithm 3. Similarly to the previous two, this algorithm also revolves around the $Harmonize()$ function but considers a different set of inputs. In this third case, the purpose is to make consumer and provider intents coherent, opening up a connection on the provider side for any harmonized request intent of the consumer that has been authorized. Algorithm 3 processes all harmonized Request intents of the consumer, i.e., the set $\mathcal{I}_{\mathcal{R}cons}^{\mathcal{H}}$, against the Request intents of the provider, i.e., $\mathcal{I}_{\mathcal{R}prov}$. If some communications are present in $\mathcal{I}_{\mathcal{R}cons}^{\mathcal{H}}$ but not in $\mathcal{I}_{\mathcal{R}prov}$, then they are added to obtain the final harmonized set of Request intents for the provider, i.e., $\mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}}$. Using the same formulation as before, this can be formulated as:

$$\mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}} \leftarrow \mathcal{I}_{\mathcal{R}prov} \cup (\mathcal{I}_{\mathcal{R}cons}^{\mathcal{H}} \setminus \mathcal{I}_{\mathcal{R}prov})$$

Input: a set of Authorization intents $\mathcal{I}_{\mathcal{A}} = (\mathcal{I}_{\mathcal{A}}^{d}, \mathcal{I}_{\mathcal{A}}^{m})$, a set of Request intents $\mathcal{I}_{\mathcal{R}}$
Output: the harmonized set of request intents $\mathcal{I}_{\mathcal{R}}^{\mathcal{H}}$
1: $\mathcal{I}_{\mathcal{R}}^{\mathcal{H}} \leftarrow \mathcal{I}_{\mathcal{R}}$
2: **for** $i_a \in \mathcal{I}_{\mathcal{A}}^{m} = \{i_1, ..., i_r\}$ **do**
3: $\quad \mathcal{I}_{\mathcal{R}}^{\mathcal{H}} \leftarrow \mathcal{I}_{\mathcal{R}}^{\mathcal{H}} \cup$ HARMONIZE$(i_a, \mathcal{I}_{\mathcal{R}})$
4: **end for**
5: **return** $\mathcal{I}_{\mathcal{R}}^{\mathcal{H}}$

Algorithm 2 Algorithm for harmonization phase (discordances of type 2)

Input: a set of Request intents $\mathcal{I}_{\mathcal{R}prov}$ defined by Provider, a set of harmonized Request intents $\mathcal{I}_{\mathcal{R}cons}^{\mathcal{H}}$ defined by Consumer
Output: the harmonized set of Request intents $\mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}}$ for the Provider
1: $\mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}} \leftarrow \mathcal{I}_{\mathcal{R}prov}$
2: **for** $i_r \in \mathcal{I}_{\mathcal{R}cons}^{\mathcal{H}} = \{i_1, ..., i_m\}$ **do**
3: $\quad \mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}} \leftarrow \mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}} \cup$ HARMONIZE$(i_r, \mathcal{I}_{\mathcal{R}prov})$
4: **end for**
5: **return** $\mathcal{I}_{\mathcal{R}prov}^{\mathcal{H}}$

Algorithm 3 Algorithm for harmonization phase (discordances of type 3)

5.2 Translation

After the sets of harmonized intents have been computed, and the consumer has selected a provider, the translation module translates the high-level intents, agnostic to the actual implementation, into the low-level configuration of the Kubernetes Network Policies required to enforce network isolation between workloads. The

complexity of translation depends on the selected CNI (e.g., not all CNI support cluster-wide Network Policy, thus requiring in some cases a one-to-many translation for the intents). For example, if there is no support from the CNI, intents that are global to the cluster cannot be simply translated "one-to-one" but need to be enriched with external data gathered from different modules within the cluster.

In our case, we adopted the native definition of Kubernetes Network Policies [32], which does not allow to have a by-default global Policy unlike Calico's implementation [33]. The presence of this translation module is crucial for the interoperability with multiple clusters that are possibly using multiple technologies. Different versions of this translation module could adapt the solution to different vendor-specific solutions, e.g., different CNIs and Network Policy formats. Finally, this module is also responsible for the enforcement of the generated Kubernetes Network Policies by communicating with the API server of the cluster hosting the targeted resources.

Figure 6 represents an example of such workflow considering a single intent that is translated to two Network Policies, one allowing the ingress traffic at the destination from the source, and another allowing the egress traffic for the source toward the destination.

6 Correctness and Termination of the Algorithms

This section provides some arguments on the correctness of the algorithms regulating the three phases of the approach, namely verification, translation, and harmonization. However, a proper formal proof is provided only for the harmonization algorithms since those regulating the verification and translation processes are straightforward and do not require a dedicated formalization.

Indeed, the translation algorithm is slightly more complex than a mapping, associating the fields of each intent to different elements within the Kubernetes Network Policy schema. The main source of complexity in this operation derives from the possibility of having wildcard characters within the intent language. In those cases, the mapping requires the knowledge of additional information which are retrieved from the cluster's state (e.g., to translate an intent selecting all namespaces the algorithm must know which namespaces are present within the cluster). Instead, the verification algorithm iterates over all tuples composed by the provided sets of intents. Each one

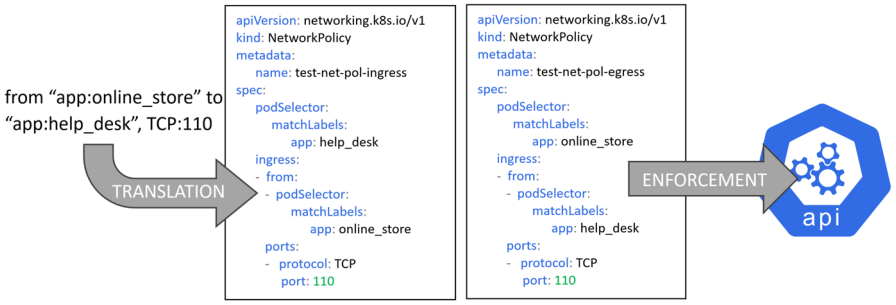


Fig. 6 Example of translation of an intent

of them is processed to look for a possible overlap and terminating in the affirmative case. The algorithm does not perform any complex set operation nor uses recursion.

6.1 Termination of Harmonization Algorithms

The core of all the presented harmonization algorithms, i.e., Algorithms 1, 2, 3, is the auxiliary function $Harmonize(i, I)$, which takes a single intent *i* and a set of intents *I* as arguments. In all the aforementioned algorithms, this function is invoked within a bounded loop. Therefore, to prove that all harmonization algorithms terminate, it is sufficient to prove the termination of the $Harmonize$ function.

We start defining a new auxiliary function μ, which takes an intent's condition as input and outputs a 4-tuple. The resulting 4-tuple is composed of a measure of each condition's element:

$$\mu(C) = (size(proto), size(p), size(src), size(dst))$$

Let the function *size*(*x*) denote the size of *x*, with the meaning depending on the type of *x*:

- if *x* is *proto*, then *size*(*x*) is the number of elements in *proto*, with $size(\{ALL\}) = size(\{STCP, UDP, TCP\}) = 3$. Possible return values are from 1 to 3, as in this last case.
- if *x* is *p*, then *size*(*x*), with $x = (p_i, p_f)$, is $p_f - p_i$. Accordingly to the definition of *p*, the result is an integer within the range 0-65535.
- if *x* is *src*, or equivalently *dst*, $size(x) = size(selector)$, where *selector* is the selector included in *x*. In case the selector is of type S_{CIDR}, then $size(S_{\text{CIDR}}) = \mathcal{BC}(S_{\text{CIDR}}) - \mathcal{NW}(S_{\text{CIDR}})$, and the result is limited to the set of IPv4 addresses. Beware that the difference is evaluated on the integer interpretation of the IP addresses, i.e., given an IP address $x_1.x_2.x_3.x_4$, its corresponding integer representation is computed as $x_1 \cdot 2^{24} + x_2 \cdot 2^{16} + x_3 \cdot 2^8 + x_4 \cdot 2^0$. Instead, in case the selector is of type S_{PodNs}, then $size(S_{\text{PodNs}}) = |\phi(S_{\text{PodNs}})|$, i.e., the cardinality of the corresponding set of Kubernetes Pods. In this case, the return value is upper bounded by the number of Pods within the cluster, which is always a finite number.

Therefore, the codomain of function μ is $\mathbb{N}^4$, which is well-founded under the lexicographical ordering operator $<_{lex}$.

Having defined this, we now prove that $Harmonize(i, D)$ terminates. In the general case, let *i.C* be the condition of the input intent. Whenever an overlapping intent $d \in D$ is found, the $Harmonize()$ function partitions the input intent's condition, i.e., *i.C*, into a finite set of fragments $\{C_1, \ldots, C_k\}$. Then $Harmonize$ is recursively called on any elements C_j in $\{C_1, \ldots, C_k\}$. By construction, we have that each condition C_j is such that $\mu(C_j) <_{lex} \mu(C)$, because each computed fragment is the result of a set intersection and difference along one of the four dimensions. As a result, at least one axis strictly decreases in size, while all others remain unchanged.

Finally, since $(\mathbb{N}^4, <_{\text{lex}})$ is well-founded, there can be no infinite descending chain

$$\mu(C) >_{\text{lex}} \mu(C_1) >_{\text{lex}} \mu(C_2) >_{\text{lex}} \cdots$$

Consequently, since $i.C$ could be fragmented up to a finite number of elements, the recursion must terminate in a finite number of steps.

6.2 Correctness of Harmonization Algorithms

As stated above, all harmonization algorithms revolve around the recursive function $Harmonize$. Therefore, it is necessary to prove its correctness to then derive the correctness of the three harmonization algorithms.

Given the function $Harmonize(i, I)$ and being I_H the set of intents generated as output, the following invariant is defined:

$$\forall j \in I, \forall h \in I_H : j.C \cap h.C = \emptyset \wedge h.C \setminus i.C = \emptyset \tag{1}$$

If Eq. (1) remains true at the end of each recursive call of the function, then we can state that the algorithm is correct. Indeed, the above formula implies that each resulting intent, $h \in I_H$ is disjoint from the input set of intent, I, and moreover, the resulting set I_H is composed of intents which condition is a partition of the input intent condition $i.C$, i.e., $h.C \setminus i.C = \emptyset$. Overall, this models the scope of the $Harmonize$ function.

The following proof will be given by contradiction. Let us suppose that the logical negation of the invariant defined in Eq. (1) holds true:

$$\exists j \in I,\ \exists h \in I_H :\ j.C \cap h.C \neq \emptyset \vee h.C \setminus i.C \neq \emptyset \tag{2}$$

If Eq. (2) is true, there are two possible scenarios which must be analyzed. The first one is that $I_H = \emptyset$, and in this case the above statement is obviously false since no h exists. The second one is that I_H is non-empty. However, this would imply that there must have been a call to $Harmonize(i, I)$ resulting in the addition of an intent h to the set of results I^H, with h being the result of a set difference between i and the elements of I. However, the only case in which an element could be added to the resulting set is within the `else` condition of Algorithm 1, line 52. In order for this code block to be executed, the variable `dirty` must be set to false. This variable is initialized to false and changed to true only if the currently processed intent i is in overlap with an element in the considered set of intents I, that is $\exists j \in I : i.C \cap j.C \neq \emptyset$. Therefore, if $h \in I_H$, this condition must never happen and the opposite must be true, i.e., $\forall j \in I :\ i.C \cap j.C = \emptyset$. However, this contradicts the first disjunct of Eq. (2). Moreover, also the second disjunct of Eq. (2), i.e., $h.C \setminus i.C \neq \emptyset$, is false by construction. Each intent in I_H is constructed by the $Harmonize$ function through the set difference operation applied to the input intent i and elements within the input set I. Therefore, since any element in I_H is generated by subtraction from i, it must be that each element $h \in I_H$ is a subset of i, and it is always true that $h.C \setminus i.C = \emptyset$. In the end, Eq. (2) has been proved to be false under the contradictory assumption that Eq. (1) is false, which means that the only possibility is for the opposite, i.e., Eq. (1), to be true.

Considering this result, the correctness proof could be extended to the correctness of the three harmonization algorithms. Starting with Algorithm 1, its goal is to remove from the set of request intents I_R all fragments overlapping with the set of denied communication intents I_A^d, producing the harmonized set I_R^H. In this case, the invariant modeling the correctness of the result is:

$$\forall j \in I_R^H, \forall k \in I_A^d : j.C \cap k.C = \emptyset \wedge \forall h \in I_R^H, \exists r \in I_R : h.C \setminus r.C = \emptyset \quad (3)$$

Looking at Algorithm 1, the initially empty set I_R^H is made up from the results of $Harmonize(i_r, I_A^d)$, with $i_r \in I_R$. Applying the previously proved invariant, the partial result produced by $Harmonize(i_r, I_A^d)$ must be disjoint from I_A^d. However, also their composition, i.e., I_H^R, must be disjoint from I_A^d. This satisfies the first part of Eq. (3). The second part is again trivially true by construction. If any element in the resulting set I_R^H is generated starting from an element in I_R, through set difference operations, then, for all elements of I_R^H, it always exists an element in I_R which corresponds to its superset, i.e., $\forall h \in I_R^H, \exists r \in I_R : h.C \setminus r.C = \emptyset$.

This proves the Eq. (3) and the correctness of Algorithm 1.

Next, the goal of Algorithm 2 is to combine two sets in an anomaly-free way, i.e., the set I_R resulting from the previous Algorithm 1 and the set of mandatory communication intents I_A^m. The resulting set I_R^H is the union of all elements of I_R and all elements of I_A^m. However, because of the anomaly-free requirements, all intents in I_R^H must be added only once to avoid redundancy. Its correctness may be formulated as the following invariant:

$$(\forall h \in I_R^H, \exists k \in (I_R \cup I_A^m) : h.C \setminus k.C = \emptyset) \wedge (\forall i_1, i_2 \in I_R^H : i_1.C \cap i_2 = \emptyset) \quad (4)$$

The first part is trivial. Every element in I_R^H is either an element of I_R, added at the beginning of Algorithm 2, or the result of $Harmonize(i_a, I_R)$, with $i_a \in I_A^m$. The result of which is an element of I_A^m processed through zero or more set difference operations with elements of I_R, i.e., any result of $Harmonize(i_a, I_R)$ is a subset of an element in I_A^m. For the second part of Eq. (4) it could be proved using the correctness of $Harmonize$ which was proved using Eq. (1). Considering as a hypothesis that I_R is anomaly-free, and having proved that the set produced by $Harmonize(i_a, I_R)$ is disjoint from I_R, then also the statement $\forall i_1, i_2 \in I_R^H : i_1.C \cap i_2.C = \emptyset$ is true. Overall, this proves that Eq. (4) holds true and that Algorithm 2 is also correct.

Finally, the case of Algorithm 3 is very similar to the previous one, as its scope is to add in an anomaly-free way all elements of the harmonized consumer request intents $I_{R\,cons}^H$ to the set of provider request intents I_{Rprov}. The correctness of the resulting intent $I_{R\,prov}^H$ is modeled by the following:

$$\begin{aligned}(\forall i \in I_{R\,prov}^H, \exists j \in (I_{R\,cons}^H \cup I_{Rprov}) : i.C \setminus j.C = \emptyset) \\ \wedge \forall (i_1, i_2 \in I_{R\,prov}^H : i_1.C \cap i_2.C = \emptyset)\end{aligned} \quad (5)$$

The proof is identical to the previous one. Specifically, the first part of Eq. (5) holds true by construction of $I_{R\,prov}^H$. Instead, the second part is true because any intent in

$I^H_{R\ prov}$ is either part of I_{Rprov}, which is supposed to be anomaly-free by hypothesis, or an intent resulting from $Harmonize(i_r, I_{Rprov})$, which was logically proved to be non overlapping with I_{Rprov}. Lastly, Eq. (5) must be true and Algorithm 3 is correct.

7 Implementation and Validation

The proposed approach has been implemented as a proof-of-concept Java program and the code has been made publicly available on GitHub[2]. Using the official Kubernetes Java SDK,[3] the program implementing the secure border agent has been deployed as a Kubernetes Controller. The agent authenticates itself and communicates with the Kubernetes API server to monitor relevant events and push the computed Network Policies. For what concerns the CNI, we implemented a translator working with Calico[4] because of its full support of the native Kubernetes Network Policy format. Moreover, the security intents are expressed using an extended version of the MSPL (i.e., Medium-level Security Policy Language) language, which is characterized by a generic syntax that abstracts the vendor-specific configuration. This choice is motivated by the fact that MSPL has already been successfully used and validated by multiple European research projects, e.g., ANASTACIA and SECURED, and research papers [34, 35].

This implementation has been validated both qualitatively with different scenarios and sets of user-defined intents in the context of the EU project FLUIDOS, and quantitatively with performance tests. On the one hand, Sect. 7.1 presents a representative and comprehensive use case as validation of the work. Since the verification strategy defined in the current work is very simple, this part is not shown within the use case, as it was also presented with an exhaustive example in Sect. 5.1.1. Instead, a partial overlap between user and consumer intents is considered in order to properly showcase all features of the harmonization, which is the most complex operation of the whole approach. On the other hand, Sect. 7.2 presents the results of quantitative tests executed to assess the performance of all three steps of the proposed solution, i.e., verification, harmonization, and translation.

7.1 Qualitative Validation on Use Case Scenario

The scenario considered in the use case is shown in Fig. 7. It is characterized by two domains, represented by the cluster with a blue border and the one with an orange border, each hosting a different service, composed of multiple applications. The first cluster hosts a simplified e-commerce service composed of four applications. The second one hosts a warehouse management service composed of two applications, and a monitoring agent to keep track of cluster events and application-specific logs.

[2] https://github.com/netgroup-polito/secure-border-controller.

[3] https://github.com/kubernetes-client/java.

[4] https://github.com/projectcalico/calico.

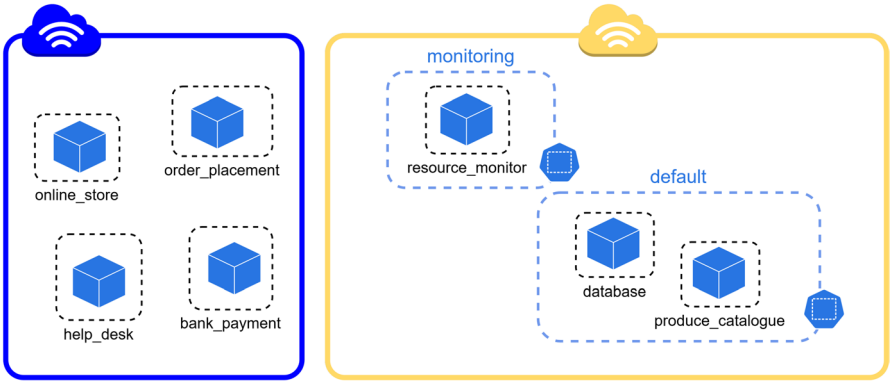


Fig. 7 Example scenario

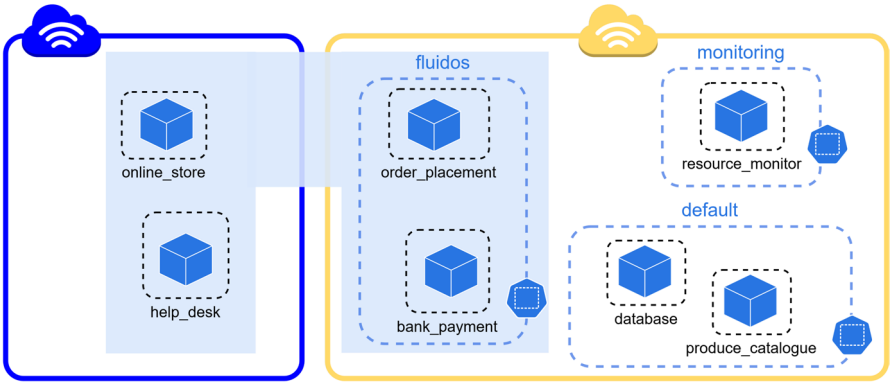


Fig. 8 Example scenario after offloading

All the resources are labeled at the application level as represented within the image, which means that all pods running the same application will have the same label.

In this scenario, the user of the blue cluster wants to use an external warehouse service, which is offered by the user of the orange cluster. To achieve this, the consumer will initiate a peering with the provider and offload parts of its workloads to the remote cluster, as shown in Fig. 8. The goal is to automatically enforce isolation between the two tenants over the border of the virtual cluster during the resource acquisition process. Note that the virtual cluster is represented in the figure with a light blue box expanding from the physical cluster of the consumer to the remote one of the provider.

At the same time, some connections should be opened for different reasons:

- The consumer wants to use the Internet connection of the hosting cluster for the application handling bank payments, which must be able to communicate with the external bank's payment network;
- The consumer wants to limit the communications with the provider's service, i.e.,

warehouse management service, only to a specific application, i.e., order placement, blocking all the other communications;
- The provider wants that each application running in its cluster, even the guest ones, can be contacted by the monitoring agent to retrieve application's logs for security reasons.

All the other communications are, by default, blocked, following the least privilege principle. Note that for the consumer only the Private and Request intents are relevant, whereas for the provider only the Authorization and Request ones. The complete set of intents defined by both parties is presented in a simplified form in Table 4.

Starting from the initial situation in Fig. 7, the consumer searches for possible providers offering the desired resource by relying on the underlying computing continuum provider. The offers of all possible providers are collected, and the consumer has to pick one of them (this is done with the verification process described in Sect. 5.1.1). After that, the peering is initiated, and the intents need to be harmonized, following the sequential algorithms presented in Sect. 5.1.2. Focusing on the harmonization, the first input is the set of Request intents of the consumer, composed in this case of just two elements:

1. Allow traffic from $app : order_placement$ to all destinations in the hosting cluster $* : *$, any port and protocol;
2. Allow traffic from $app : bank_payment$ to all IP addresses, i.e., 0.0.0.0/0, any port and protocol.

In the harmonization process, each type of discordance is solved separately. Starting from the first discordance type, the first intent overlaps with the authorizations defined by the provider, which exposes only the application $app : product_catalogue$ at port 80 TCP and blocks all the other communications. Moreover, the second intent is partially overlapping with another Authorization intent of the provider, which blocks the 0.0.0.0/4 range of addresses (Fig. 9). Continuing with the second discordance

Table 4 Intent sets in the use case

#	*src*	*dst*	*p*	*proto*
Request intents ($\mathcal{I}_{\mathcal{R}}$) of consumer				
1	$\mathcal{S}_{\text{PodNs}} = (\{(app, order_placement)\}, \{(name, fluidos)\})$	$\mathcal{S}_{\text{PodNs}} = (\{(*, *)\}, \{(name, default)\})$	$*$	$\{ALL\}$
2	$\mathcal{S}_{\text{PodNs}} = (\{(app, bank_payment)\}, \{(name, fluidos)\})$	$\mathcal{S}_{\text{CIDR}} = 0.0.0.0/0$	$*$	$\{ALL\}$
Denied communication intents ($\mathcal{I}^d_{\mathcal{A}}$) of provider				
1	$\mathcal{S}_{\text{PodNs}} = (\{(*, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/16$	$*$	{UDP}
2	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/18$	(80, 80)	{TCP}
3	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.0.0.0/8$	$*$	{ALL}
Mandatory communication intents ($\mathcal{I}^m_{\mathcal{A}}$) of provider				
1	$\mathcal{S}_{\text{PodNs}} = (\{(*, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/16$	$*$	{UDP}
2	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.192.0.0/18$	(80, 80)	{TCP}
3	$\mathcal{S}_{\text{PodNs}} = (\{(role, *)\}, \{(*, *)\})$	$\mathcal{S}_{\text{CIDR}} = 130.0.0.0/8$	$*$	{ALL}

```
[INFO]    List of harmonized Request intents (CONSUER) after type-1 resolution:
 (*) src:[ app:order_placement - name:fluidos ], sPort:[*],
     dst:[ app:product_catalogue - name:default ], dPort:[80], protocol:[TCP]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[128.0.0.0/1], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[64.0.0.0/2], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[32.0.0.0/3], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[16.0.0.0/4], dPort:[*], protocol:[ALL]
```

Fig. 9 Result of type-1 discordance resolution

```
[INFO]    List of harmonized Request intents (CONSUMER) after type-2 resolution:
 (*) src:[ app:order_placement - name:fluidos ], sPort:[*],
     dst:[ app:product_catalogue - name:default ], dPort:[80], protocol:[TCP]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[128.0.0.0/1], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[64.0.0.0/2], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[32.0.0.0/3], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[16.0.0.0/4], dPort:[*], protocol:[ALL]
 (*) src:[ app:resource_monitor - name:monitoring ], sPort:[*],
     dst:[ *:* - name:fluidos ], dPort:[43], protocol:[TCP]
```

Fig. 10 Result of type-2 discordance resolution

```
[INFO]    List of harmonized Request intents (PROVIDER) after type-3 resolution:
 (*) src:[ app:order_placement - name:fluidos ], sPort:[*],
     dst:[ app:product_catalogue - name:default ], dPort:[80], protocol:[TCP]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[128.0.0.0/1], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[64.0.0.0/2], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[32.0.0.0/3], dPort:[*], protocol:[ALL]
 (*) src:[ app:bank_payment - name:fluidos ], sPort:[*],
     dst:[16.0.0.0/4], dPort:[*], protocol:[ALL]
 (*) src:[ app:resource_monitor - name:monitoring ], sPort:[*],
     dst:[ *:* - name:fluidos ], dPort:[43], protocol:[TCP]
```

Fig. 11 Result of type-3 discordance resolution

type, the provider defined only one intent within the mandatory list of communications. Moreover, that intent describes a communication that has no intersection with the ones in the current set of the consumer's harmonized Request intents. Consequently, the mandatory communication is added to the final harmonized set with no modification (Fig. 10). Then, concerning the third discordance type, as initially the set of the provider's Request intents is empty, the consumer's Request intents are simply opportunely modified and added to the provider's set because there is no possible overlap (Fig. 11).

After having performed the harmonization procedure for all the involved sets, the result is passed to the translator module. This component processes each intent and

generates one or more Kubernetes Network Policies, depending on the complexity involved in the translation. Finally, the Network Policies are applied to enforce the isolation as soon as the applications are offloaded to the remote cluster. Taking as example just the first harmonized Request intent, i.e., allow communication from $app : order_placement$ to $app : product_catalogue$ at port 80 TCP, its translation into a Network Policy is shown in Fig. 12. In order to carry out this operation, the translation process has to map all fields of the intent within the predefined structure of the Kubernetes Network Policy. Depending on the intent, the translation process could result into one or more Network Policies. For instance, an intent describing a communication between two pods, such as the one considered here, requires the creation of two Network Policies: a first one to allow the egress connection on the source pod to the destination pod (left portion of Fig. 12), and a second one to allow the ingress communication on the destination pod from the source pod (right portion of Fig. 12). Instead, if the intent describes a communication between a pod and an external entity, i.e., an IP range, a single network policy allowing the egress connection on the pod towards the external entity would be sufficient. Finally, in Kubernetes, Network Policy are namespaced objects, i.e., they are defined within a namespace, but the presented intent language allows the definition of cluster-wide communications using the wildcard symbol "*". Such intents are mapped to multiple network policies, as many as the namespaces present within the cluster. This file is then pushed to the API server of the provider cluster for the actual enforcement.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: request_1_egress
  namespace: fluidos
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: order_placement
  egress:
    to:
    - namespaceSelector:
        matchLabels:
          name: default
      podSelector:
        matchLabels:
          app: product-catalogue
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: request_1_ingress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: product_catalogue
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: fluidos
      podSelector:
        matchLabels:
          app: order_placement
    ports:
    - port: 80
      protocol: TCP
```

Fig. 12 Resulting network policy

7.2 Quantitative Evaluation

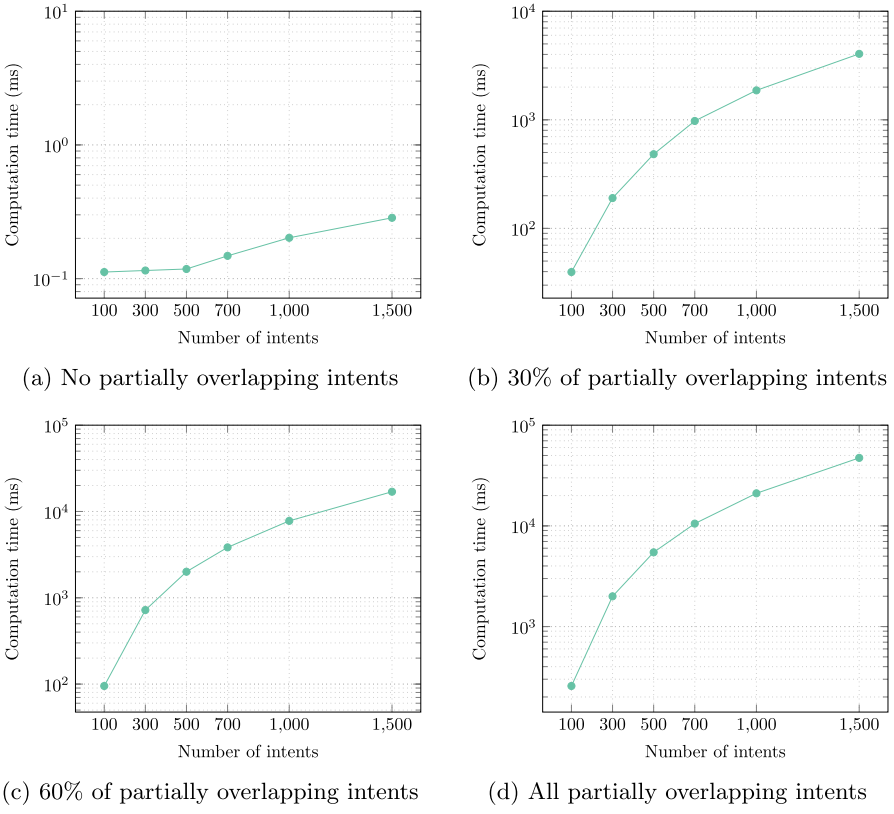
The tests have been conducted using synthetically generated intents and a computer equipped with an Intel Ultra7 155 H and 32 GB of internal memory. For all test cases, several runs have been executed, and their computation time has been recorded in nanoseconds. The conducted quantitative evaluation aims to assess the performance of the approach by analyzing the performance of the three core phases, i.e., harmonization, verification, and translation.

Starting with the harmonization, this phase has been extensively assessed as being by far the most complex among the three. Specifically, the tests do not focus on the complete harmonization phase nor on its key components, i.e., Algorithms 1, 2, and 3. To perform a more fine-grained analysis, general to all the aforementioned algorithms, the analysis considers the performance of the $Harmonize$ function. For each run, the function has been applied to a single intent and a set of intents with a varying dimension to evaluate the scalability of the harmonization phase concerning the number of intents. Moreover, to evaluate its scalability with respect to the complexity of intents, different cases have been considered. Specifically, the test cases cover increasing percentages of overlap between the single intent given in input and the set of intents, e.g., if $\mathcal{I}$ has 100 intents and the case with 30% overlap is considered, then 30 intents are partially overlapping with the input intent i while the other 70 are disjoint from i. These different test cases have a direct impact on the computation time, because if an overlap is detected, the set difference is computed and function $Harmonize$ is recursively called zero or multiple times. Instead, if there is no overlap, the intent is added to the resulting set without recursion or possibly complex set operations.

The results are represented in Fig. 13, which contains four charts corresponding to four different percentages of overlapping intents: 0%, 30%, 60%, and 100%. For each one of these use cases, the $Harmonize$ function has been applied to intent sets with an increasing number of elements, from 100 up to 1500. Figure 13 shows, for each combination of use case and dimension, the average computation time in milliseconds over 10 different runs.

From the reported values, it appears evident that there is a correlation between the percentage of overlap and the total computation time. In the absence of any overlap between intents (Fig. 13a), the execution of $Harmonize$ effectively skips costly recursive calls and set operations. Consequently, the computation time remains extremely low and scales almost linearly with the number of intents. The higher average time recorded in this case is 0.285 ms with 1500 intents, which is negligible. Instead, if the percentage of partial overlap increases, the average computation time increases accordingly. With a higher percentage of overlapping intents, the recursive calls are triggered on a higher percentage of input data, resulting in a higher number of recursive calls and set operations. Nonetheless, the scaling is still manageable. The maximum average computation time for the worst scenario, i.e., 1500 intents, is around 4 s for the case with 30% of overlapping intents (Fig. 13b), around 17 s with 60% of overlap (Fig. 13c), and about 47 s when all intents are overlapping (Fig. 13d).

This last scenario presents some extreme characteristics that have been synthetically generated but are rarely encountered in real scenarios. However, the perfor-



(a) No partially overlapping intents

(b) 30% of partially overlapping intents

(c) 60% of partially overlapping intents

(d) All partially overlapping intents

Fig. 13 Scalability of harmonize function to varying size and complexity

mance are still acceptable given that all harmonization algorithms are parallelizable. Each one of them processes two intent sets, i.e., $\mathcal{I}_1, \mathcal{I}_2$, by looping on all elements of $\mathcal{I}_1$ and calling the recursive function as $Harmonize(i, \mathcal{I}_2)$. Since the operations of function $Harmonize$ on each intent $i \in \mathcal{I}_1$ are independent from any other element of the same set $\mathcal{I}_1$, the processing done in each loop could be parallelized. Beware that each algorithm could be parallelized, but the overall harmonization phase requires that Algorithms 1, 2, and 3 are executed sequentially. Therefore, even with parallelization, in the worst case, the harmonization performance is triple the performance of $Harmonize$.

The second analyzed phase is the verification. In this case, the verification performance has been analyzed considering single tuples composed of one Request intent and one Authorization intent. This granularity would allow the derivation of the scalability of the verification phase by linearly multiplying the results by the number of considered tuples. Remember that the verification works on a set of Request intents $\mathcal{I}_\mathcal{R}$ and a set of Authorization intents $\mathcal{I}_\mathcal{A}$, processing each possible tuple. Therefore, evaluating the performance for verifying a single tuple is sufficient, because the scalability to multiple intents can be derived from it.

Given these assumptions, different test cases have been considered with increasing complexity of verification. In the simplest cases, no overlap is presented between the tuple and the verification terminates once this is detected. This check might require a different computation time depending on the complexity of the provided intents, which could present the non-overlapping condition along different dimensions. Depending on this, the algorithm could take a different time to terminate. Moreover, the usage of one type of selector or another for source and destination results in the execution of a different branch of the algorithm, with consequently different execution times.

The specific scenarios that have been evaluated are grouped in Table 5. Each line describes a different scenario, and the verification complexity increases from top to bottom. For each one of them, 20 different runs have been performed, and the average computation is reported in the second column of Table 5. The results showcase a consistently fast computation time across all scenarios, which always remains well below 0.05 ms for each considered tuple. This confirms the minimal impact of the verification phase with respect to the harmonization one. Having a fast and dedicated verification algorithm enables the scalability and applicability of the presented approach to multiple providers. Finally, the verification of each tuple is self-contained and could be done in isolation with respect to the others, enabling the parallelization of the verification algorithm.

Lastly, to evaluate the performance of the implemented translation algorithm, three different complexity scenarios are considered:

- With exclusive usage of S_{CIDR} selectors for both source and destination;
- With exclusive usage of S_{PodNs} selectors for both source and destination;
- With mixed usage of S_{CIDR} and S_{PodNs} selectors for both source and destination.

For all the other intent fields, i.e., protocol type and port, the translation implies a trivial mapping into the correct field of the Kubernetes Network Policy schema. For

Table 5 Performance analysis of verification function with varying complexity

Test scenario description	Avg. computation time (ms)
Incompatible parameter (accept monitoring)	0.0030435
Single non-overlapping field (protocol)	0.0403948
Single non-overlapping field (port)	0.0414726
Single non-overlapping field (src, with S_{CIDR})	0.0393982
Single non-overlapping field (src, with S_{PodNs})	0.0313571
Single non-overlapping field (dst, with S_{CIDR})	0.0428547
Single non-overlapping field (dst, with S_{PodNs})	0.0280475
Single non-overlapping field (with different selector types)	0.0274011
All overlapping fields (srd and dst with S_{CIDR})	0.0358931
All overlapping fields (srd and dst with S_{PodNs})	0.0266328
All overlapping fields (srd and dst with mixed selectors)	0.0349272

Table 6 Performance analysis of translation function

Test scenario description	Avg. computation time (ms)
Exclusive usage of S_{CIDR} for both src and dst	0.0030435
Exclusive usage of S_{PodNs} for both src and dst	0.0403948
Mixed usage of S_{CIDR} and S_{PodNs} for src and dst	0.0414726

this reason, their contribution to the complexity is not considered within the test scenarios.

The performance has been evaluated over 20 runs for each scenario, and the resulting average computation time is reported in Table 6. Despite the growing complexity, the results highlight that the algorithm exhibits consistent performance across all scenarios, with all average execution times remaining bounded to very low values that could be considered negligible.

8 Conclusion and Future Work

This paper introduced an intent-driven security solution designed to enforce network isolation within the cloud computing continuum. By leveraging user-defined intents, the approach streamlines the automated enforcement of network isolation primitives, i.e., Kubernetes Network Policies, across multiple devices. By incorporating a structured process of verification, harmonization, and translation, our solution effectively implements multi-tenant network security isolation in the continuum. The approach has been implemented and validated on possible use cases, confirming its applicability to the presented problem.

Future work will focus on refining the capabilities of the approach proposed in this paper. This includes the definition of a proper verification strategy to rank the providers based on intents' alignment, and the integration of service mesh technologies to cover the whole spectrum of communications within the continuum. Finally, the integration of dynamic adjustments to the security policies could open new research opportunities in the design of adaptive security solutions based on workload behavior. For instance, the proposed approach may be extended to provide a dynamic lifecycle, where consumers can decide to release the resources borrowed from a provider and go back to the verification stage to select more appropriate providers.

Acknowledgements This work was partly supported by EU Horizon project FLUIDOS, under grant agreement 101070473.

Author Contributions All authors contributed equally to this work.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no Conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Enciso, A.R., Murcia, J.M.B., Zarca, A.M., Skarmeta-Gómez, A.F.: Dynamic multi-method allocation for intent-based security orchestration. J. Netw. Syst. Manag. **33**(1), 21 (2025). https://doi.org/10.1007/S10922-024-09896-8
2. Zambianco, M., Cretti, S., Siracusa, D.: Cost minimization in multi-cloud systems with runtime microservice re-orchestration. In: Proc. of the 27th Conference on Innovation in Clouds, Internet and Networks, ICIN 2024, Paris, France, March 11-14, 2024, pp. 65–72 (2024). https://doi.org/10.1109/ICIN60470.2024.10494463
3. Ullah, A., Kiss, T., Kovács, J., Tusa, F., Deslauriers, J., Dagdeviren, H., Arjun, R., Hamzeh, H.: Orchestration in the cloud-to-things compute continuum: taxonomy, survey and future directions. J. Cloud Comput. **12**(1), 135 (2023). https://doi.org/10.1186/S13677-023-00516-5
4. Cai, F., Zhu, N., He, J., Mu, P., Li, W., Yu, Y.: Survey of access control models and technologies for cloud computing. Clust. Comput. **22**(Supplement), 6111–6122 (2019). https://doi.org/10.1007/S10586-018-1850-7
5. Wang, X., Du, J., Liu, H.: Performance and isolation analysis of runc, gvisor and kata containers runtimes. Clust. Comput. **25**(2), 1497–1513 (2022). https://doi.org/10.1007/S10586-021-03517-8
6. Wiz: The 2023 Kubernetes Security Report. Available: https://www.wiz.io/lp/the-2023-kubernetes-security-report, Visited: 2025-02-28
7. Minna, F., Blaise, A., Rebecchi, F., Chandrasekaran, B., Massacci, F.: Understanding the security implications of Kubernetes networking. IEEE Secur. Priv. **19**(5), 46–56 (2021). https://doi.org/10.1109/MSEC.2021.3094726
8. Budigiri, G., Baumann, C., Mühlberg, J.T., Truyen, E., Joosen, W.: Network policies in kubernetes: Performance evaluation and security analysis. In: Joint European Conference on Networks and Communications & 6G Summit, EuCNC/6G Summit 2021, Porto, Portugal, June 8-11, 2021, pp. 407–412 (2021). https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482526
9. Pizzato, F., Bringhenti, D., Sisto, R., Valenza, F.: An intent-based solution for network isolation in Kubernetes. In: 10th IEEE International Conference on Network Softwarization, NetSoft 2024, Saint Louis, MO, USA, June 24-28, 2024, pp. 381–386 (2024). https://doi.org/10.1109/NetSoft60951.2024.10588939
10. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F.: Automation for network security configuration: state of the art and research trends. ACM Comput. Surv. **56**(3), 57–15737 (2024). https://doi.org/10.1145/3616401
11. Bringhenti, D., Sisto, R., Valenza, F.: A novel abstraction for security configuration in virtual networks. Comput. Netw. **228**, 109745 (2023). https://doi.org/10.1016/j.comnet.2023.109745
12. Scheid, E.J., Machado, C.C., Franco, M.F., Santos, R.L., Pfitscher, R.J., Filho, A.E.S., Granville, L.Z.: Inspire: Integrated nfv-based intent refinement environment. In: Proc. of the IFIP/IEEE Symp. on Integrated Network and Service Management (IM17) (2017). https://doi.org/10.23919/INM.2017.7987279

13. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: Introducing programmability and automation in the synthesis of virtual firewall rules. In: Proc. of 6th IEEE Conference on Network Softwarization, NetSoft 2020, Ghent, Belgium, June 29–July 3, 2020, pp. 473–478 (2020). https://doi.org/10.1109/NetSoft48620.2020.9165434
14. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: Automated firewall configuration in virtual networks. IEEE Trans. Depend. Secur. Comput. **20**(2), 1559 (2023). https://doi.org/10.1109/TDSC.2022.3160293
15. Szyrkowiec, T., Santuari, M., Chamania, M., Siracusa, D., Autenrieth, A., López, V., Cho, J.Y., Kellerer, W.: Automatic intent-based secure service creation through a multilayer SDN network orchestration. J. Opt. Commun. Netw. **10**(4), 289–297 (2018). https://doi.org/10.1364/jocn.10.000289
16. Cauli, C., Li, M., Piterman, N., Tkachuk, O.: Pre-deployment security assessment for cloud services through semantic reasoning. In: Proc. of Computer Aided Verification (CAV), Springer - 33rd International Conference, Virtual Event, July 20–23, 2021. Lecture Notes in Computer Science, vol. 12759, pp. 767–780 (2021). https://doi.org/10.1007/978-3-030-81685-8_36
17. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K.S., Rungta, N., Tkachuk, O., Varming, C.: Semantic-based automated reasoning for AWS access policies using SMT. In: 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018, pp. 1–9 (2018). https://doi.org/10.23919/FMCAD.2018.8602994
18. Backes, J., Bayless, S., Cook, B., Dodge, C., Gacek, A., Hu, A.J., Kahsai, T., Kocik, B., Kotelnikov, E., Kukovec, J., McLaughlin, S., Reed, J., Rungta, N., Sizemore, J., Stalzer, M.A., Srinivasan, P., Subotic, P., Varming, C., Whaley, B.: Reachability analysis for aws-based networks. In: Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 231–241 (2019). https://doi.org/10.1007/978-3-030-25543-5_14
19. Blaise, A., Rebecchi, F.: Stay at the helm: secure Kubernetes deployments via graph generation and attack reconstruction. In: IEEE 15th Int. Conf. on Cloud Computing, Barcelona, Spain, July 10–16, 2022, pp. 59–69 (2022). https://doi.org/10.1109/CLOUD55607.2022.00022
20. Minna, F., Massacci, F., Tuma, K.: Towards a security stress-test for cloud configurations. In: IEEE 15th Int. Conf. on Cloud Computing, Barcelona, Spain, July 10–16, 2022, pp. 191–196 (2022). https://doi.org/10.1109/CLOUD55607.2022.00038
21. Zhu, H., Gehrmann, C.: Kub-Sec, an automatic Kubernetes cluster AppArmor profile generation engine. In: 14th International Conference on Communication Systems & Networks, COMSNETS 2022, Bangalore, India, Jan 4–8, 2022, pp. 129–137 (2022). https://doi.org/10.1109/COMSNETS53615.2022.9668504
22. Haque, M.U., Kholoosi, M.M., Babar, M.A.: KGSecConfig: A knowledge graph based approach for secured container orchestrator configuration. In: IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15–18, 2022, pp. 420–431 (2022). https://doi.org/10.1109/SANER53432.2022.00057
23. Li, X., Chen, Y., Lin, Z., Wang, X., Chen, J.H.: Automatic policy generation for inter-service access control of microservices. In: 30th USENIX Security Symposium, USENIX Security 2021, August 11–13, 2021, pp. 3971–3988 (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/li-xing
24. EU Project: Flexible, scaLable and secUre decentralizeD Operating System (FLUIDOS). [Online]. Available: https://www.fluidos.eu/ (2022–2025)
25. EU Project: Towards a functional continuum operating system (ICOS). [Online]. Available: https://icos-project.eu/ (2022–2025)
26. EU Project: Autonomous, scalablE, tRusthworthy, intelligent European meta operating system for IoT edge-cloud continuum (aerOS). [Online]. Available: https://aeros-project.eu/ (2022–2025)
27. EU Project: A meta operating system for brokering hyper-distributes applications on cloud computing continuum (NebulOuS). [Online]. Available: https://nebulouscloud.eu/ (2022–2025)
28. EU Project: Next Generation Meta Operating Sytem (NEMO). [Online]. Available: https://meta-os.eu/ (2022–2025)
29. EU Project: A lightweight software stack and synergetic meta-orchestration framework for the next generation compute continuum (NEPHELE). [Online]. Available: https://nephele-project.eu/ (2022–2025)
30. Task Force 3: Architecture: Developing a Reference Architecture for the Continuum - Concept, Taxonomy and Building Blocks. Zenodo (2023). https://doi.org/10.5281/zenodo.8403593
31. Liqo Available: https://liqo.io, Visited: 2024-03-27

32. Documentation Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/, Visited: 2025-02-28
33. Calico Available: https://tigera.io/project-calico, Visited: 2025-02-28
34. Bringhenti, D., Valenza, F., Basile, C.: Toward cybersecurity personalization in smart homes. IEEE Secur. Priv. **20**(1), 45–53 (2022). https://doi.org/10.1109/MSEC.2021.3117471
35. Zarca, A.M., Bagaa, M., Bernabé, J.B., Taleb, T., Skarmeta, A.F.: Semantic-aware security orchestration in sdn/nfv-enabled iot systems. Sensors **20**(13), 3622 (2020). https://doi.org/10.3390/s20133622

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Francesco Pizzato received his M.Sc. degree in computer engineering from Politecnico di Torino, Italy, in 2023, where he is currently pursuing his Ph.D. degree in control and computer engineering. His research interests include novel network and cloud technologies, automatic orchestration and configuration of security functions, and cloud security.

Daniele Bringhenti received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2019 and 2022 respectively, where he is currently a fixed-term Assistant Professor. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, formal verification of network security policies.

Riccardo Sisto received the Ph.D. degree in Computer Engineering in 1992, from Politecnico di Torino, Italy. Since 2004, he is Full Professor of Computer Engineering at Politecnico di Torino. His main research interests are in the area of formal methods, applied to distributed software and communication protocol engineering, distributed systems, and computer security. He has authored and co-authored more than 100 scientific papers. He is a Senior Member of the ACM.

Fulvio Valenza received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively, where he is currently an Associate Professor. His research activity focuses on network security policies, orchestration and management of network security functions in SDN/NFV-based networks, and threat modeling.

Authors and Affiliations

Francesco Pizzato[1] · Daniele Bringhenti[1] · Riccardo Sisto[1] · Fulvio Valenza[1]

✉ Francesco Pizzato
francesco.pizzato@polito.it

Daniele Bringhenti
daniele.bringhenti@polito.it

Riccardo Sisto
riccardo.sisto@polito.it

Fulvio Valenza
fulvio.valenza@polito.it

[1] Dip. Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Turin, Italy