Empirical Software Engineering manuscript No.
(will be inserted by the editor)

On Practitioners' Concerns when Adopting Service Mesh Frameworks

Yihao Chen · Eduardo Fernandes · Bram Adams · Ahmed E. Hassan

Received: date / Accepted: date

Abstract *Context:* The emerging service mesh architecture tries to simplify microservices by delegating crucial tasks to dedicated infrastructure. However, service mesh introduces new notions and enables complex capabilities such as sidecar proxies that inevitably bring major adoption concerns. *Objective:* We investigate the adoption concerns in two dominant open-source service mesh frameworks via a mixed-methods empirical investigation of the past, current and evolution of 5,497 practitioner questions posted on generic and framework-specific question-and-answer fora. *Method:* We first mine the topics of questions with the help of Dynamic Topic Modeling (DTM). We identify evolution by applying topic modelling to time periods and aggregating topics into macro-topics. We conduct a qualitative analysis to understand the three major types of questions and to generalize common fix patterns for the extracted error symptoms. We consulted a service mesh domain expert to provide feedback on our findings and discuss implications. *Results:* We found that about half of the questions are error-related and mined 18 topics, covering service mesh traffic, infrastructure, security, observability and application. We discovered a drastic decline in traffic-related concerns while finding persisting infrastructure-related concerns and a rise in security and observ-

Yihao Chen
School of Computing, Queen's University, Kingston, ON, Canada
E-mail: yihao.chen@cs.queensu.ca

E. Fernandes
School of Computing, Queen's University, Kingston, ON, Canada
E-mail: eduardo.fernandes@queensu.ca

B. Adams
School of Computing, Queen's University, Kingston, ON, Canada
E-mail: bram.adams@queensu.ca

A.E. Hassan
School of Computing, Queen's University, Kingston, ON, Canada
E-mail: hassan@queensu.ca

ability concerns. We identified 54 error symptoms from two popular service mesh frameworks and generalized 9 common fix patterns. We found complex symptom-to-fix relationships, yet, surprisingly, minimal configuration changes were able to fix most symptoms. *Conclusion:* Providing consistent documentation and practical automation that assists customization of service mesh deployment and functionalities is crucial in the current service mesh domain, given the diversity of discovered intentions, goals and symptoms. Furthermore, there should be more work towards better container orchestration to deploy service mesh frameworks and reliable customization of security and observability service mesh features.

Keywords Service mesh · Microservices · Mixed-methods empirical study · Discussion forum · Technology adoption · Software architecture

1 Introduction

Microservices is an architectural pattern for heterogeneous applications that gained popularity in recent years [19]. Microservices allow distributed development and deployment of software components that together form a large application. In a traditional microservices architecture, software components handling the communication between services are known as shared Software Development Kits (SDKs). With the help of SDKs, services written in different languages and frameworks follow the same standards to keep their functionalities consistent throughout a microservices network[1].

However, despite their wide adoption, traditional microservices architectures that rely on language and framework-specific SDKs [30]. Custom-built SDKs that suit large projects would require careful development efforts and are prone to inconsistencies due to human errors. For example, if one would like to enforce a security measure in a microservices network, the developers must carefully maintain and release multiple SDKs across their microservices to avoid security vulnerabilities led by SDK inconsistency [20]. In addition to the security cost, releasing SDKs into thousands of microservices in the production environment also requires sophisticated control since different versions of SDKs may not correctly communicate with each other.

One of the major technologies trying to reduce these costs are so-called service meshes. A service mesh is a dedicated infrastructure layer that deploys and manages a list of proxies alongside each service to intercept network communication [30]. By doing so, a service mesh network could monitor and manipulate traffic towards large deployments of heterogeneous services just by making configurations changes, without changing the source code of said services. Such proxies essentially eliminate the maintenance overhead of SDKs [28] and promote the reuse of microservices. According to the 2022 Annual APIs and Integration Report [10], over 40% of organizations with a microservices application implement service mesh, although the same report indicates that

[1] `https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh`

only 9% of those organizations' respondents claim they face "no challenges" while adopting service mesh frameworks.

As an emerging technology, service mesh may not suffice to address the adoption concerns in the microservices domain or may even lead to new critical concerns regarding their usability, performance, and runtime stability [30]. For instance, service mesh proxies require dedicated management effort and are prone to occasional failure. Additionally, previous work [24] suggests that advanced features in a service mesh framework could confuse new adopters. Unfortunately, to our knowledge, existing empirical studies [22, 24, 37] on service mesh primarily focus on controlled experiments regarding framework-specific functionalities, such as the exploration of optimal traffic control configurations. Thus, empirical knowledge on service mesh adoption concerns remains scarce, if not non-existent.

This paper introduces a mixed-method empirical study of 5,497 posts on Stack Overflow and on public discussion fora hosted by open-source service mesh frameworks. Our goal is to understand practitioners' concerns when adopting service mesh frameworks, with a particular focus on question types (e.g., "how-to" and "error-related"), topics (e.g., "Access Control" and "Machine Management"), error symptoms (e.g., "Certificate mismatch") and their common fixes (e.g., "Correct certificate signing"). We target open-source service mesh frameworks that previous work commonly studied [22, 24, 30, 37], namely Istio, and Consul. We validate and discuss the findings of our study with a service mesh expert who has over five years of cloud computing and open-source experience in the service mesh domain.

Regarding the study methodology, we use keyword heuristics to characterize the different types of questions and their distribution in the dataset. We then adopt Dynamic Topic Models (DTMs) [8] to mine the topics of the analyzed questions in terms of service mesh capabilities that concern the practitioners. We next apply aggregation and regression analysis to topic time frames of 3 months to reveal the evolution of topics and identify observable trends (e.g., increase or decrease of topic prevalence) over time. We compare our service mesh findings to traditional microservices in all these analyses. Finally, we conduct a qualitative analysis based on representative samples of knowledge-transfer, how-to and error-related questions to understand question intentions, adoption goals and error symptoms. For error symptoms, we additionally assess common fix patterns for each symptom.

Our main study results and their implications include the following:

- Error-related concerns are predominant and represent 48.3% of all concerns faced while adopting service mesh, against 29.9% for traditional microservices. On the other hand, service mesh practitioners post surprisingly few questions (6.3%) on the general concepts and practices of the technology, compared to 19.8% in the traditional microservices domain. Because service mesh practitioners are primarily concerned with fixing encountered error symptoms, practitioners may benefit from automated suggestions of common fix patterns for encountered errors. Unfortunately, the answer ac-

ceptance rate of 40.6% on Stack Overflow suggests a lack of expertise in the domain, which may be mitigated over the years with the increasing popularity of the service mesh frameworks.
- Service mesh, while aiming to be a dedicated layer to reduce operational overhead [44,48], seems challenging to deploy and manage. We find that most (27.4%) of the questions in the service mesh domain relate to the underlying infrastructure layer based on container orchestration platforms such as Kubernetes[2], similar to the traditional microservices domain (22.8%). On the other hand, traffic-related features unique to service mesh frameworks also attract significant concerns, while traditional microservices practitioners benefit from more mature technologies such as API gateways and load balancers, leading to less concern.
- We discover a significant decline in the prevalence of service mesh traffic-related topics representing the technology's core functionalities, such as "Traffic and Gateway" and "Routing and Services" regarding handling and directing microservices traffic. Topics on infrastructure rebounded after an initial decline and rose along with the rapid growth of studied service mesh frameworks and underlying modern container orchestration platforms. On the other hand, we discovered and verified with the interviewed service mesh expert that the rising concerns regarding service mesh security and observability could be underestimated, which should draw more research attention.
- A majority of knowledge-transfer questions (58%) in the service mesh domain are about understanding the availability of functionalities. Carried-over knowledge from traditional microservices does not necessarily translate to a good understanding of service mesh features. While both studied frameworks are feature-rich and highly configurable, there are specific areas in each framework such as external system integration, customized routing and installation/upgrading that raise most concerns and thus should draw research attention. Providing more documentation and support in these areas could help service mesh practitioners overcome frequent concerns.
- Service mesh practitioners face diverse yet hard-to-pinpoint error symptoms and unintuitive error messages. Among the 54 error symptoms we mine from Istio and Consul, we find that 39 of the symptoms appear across the frameworks and even within different areas of concern for each framework separately. Additionally, in 29 cases, we encounter complex error-to-fix relationships, such as multiple fixes being recorded for a single error symptom, while in 15 cases, a single fix could also fix many different symptoms. It is challenging for typical practitioners to follow a systematic approach to find common fix patterns to a problem due to a general lack of consistent one-on-one relationship between an error symptom and a common fix. We encourage framework designers to enhance the error messages to facilitate smoother adoption.

[2] `https://kubernetes.io/`

- Minimal modifications in configuration manifests could fix considerable error symptoms that service mesh practitioners face. For example, modifying a simple port name according to the naming convention of Istio could fix a symptom of a defective security policy, which does not provide any error message that links to a port naming problem. We encourage researchers to explore approaches that could recommend proper configurations and examples based on unique user requirements.
- We found abundant knowledge resources in question answers that could serve as hints for fixing or preventing the analyzed error symptoms. Yet overall, we observed (I): a lack of reliable and well-documented practices in configuring service mesh systems linked to user requirements; (II): a lack of effective error diagnosis mechanisms, which render the knowledge resources less useful since practitioners cannot effectively use the knowledge within such resources to solve unclear problems.
- We provide the replication package of our study with the main study artifacts, including the data collection scripts, data processing and filtering scripts, topic models and data spreadsheets [15].

We organize the remainder of this work as follows. Section 2 introduces the core concepts that form and differentiate a service mesh network from traditional microservices architecture. Section 3 motivates our study with specific research questions to be answered. We also follow each detailed step with analysis protocols and data collection techniques. Sections 4 to 6 discuss the evaluation and results discussions for each research question. Section 7 summarizes our key findings and further gives actionable results to researchers and practitioners in the field. Section 8 acknowledges and presents the potential threats to our work and justifies the methodologies we adopted to best complement some of the shortcomings. Section 9 goes through previous works that relate to ours and discusses further insights with understandings from our study. Finally, Section 10 concludes our findings and looks into promising research focuses in the future.

2 Background

Microservices are designed for large-scale deployments with complex network conditions, involving the distribution of heterogeneous services across networks with numerous interactions between said services [2]. However, with the ever-increasing requirement for user experience, large organizations face increased costs to support millions of service instances deployed around the globe[3]. Since each service is managed by an individual team, any decision to release a new version of one service could have a ripple effect throughout the ecosystem of deployed microservices.

As a variant of the microservices architecture, service mesh implements the network infrastructure used by microservices to communicate with each other

[3] `https://www.youtube.com/watch?v=CZ3wIuvmHeM`

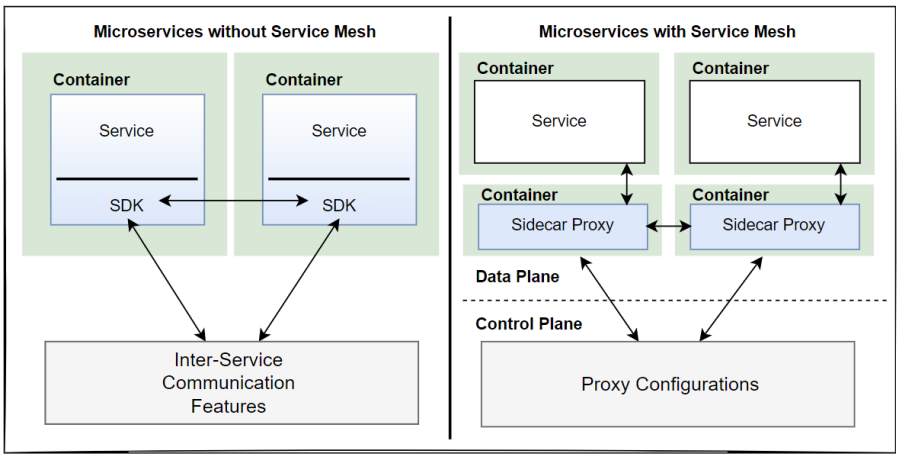

Fig. 1 Architectural Differences between Traditional Microservices and Service Mesh

differently. Previous works [24,30,43] define service mesh as a dedicated layer of infrastructure that manages the communication of microservices over heterogeneous networks. Essentially, service mesh separates microservices development from operational tasks such as deployment and monitoring, no longer required to physically alter the source code of a service just to enable debugging or other observability tasks.

Figure 1 presents an architectural comparison between a traditional microservices architecture based on shared Software Development Kits (SDKs) for inter-service communication and a typical service mesh architecture based on Istio. The left figure shows a tight relationship between individual services with a shared SDK that implements the inter-service communication features, for example, traffic monitoring. The service and the SDK are packaged within the same container, meaning minor modifications to either part would inevitably lead to a rebuild and redeployment, even if they were not meant to impact each other. Furthermore, once a service uses a given SDK, its implementation is locked into that SDK, complicating the reuse of the microservices in different settings.

On the other hand, microservices with service mesh address the above problem by adopting the "Sidecar Pattern". A sidecar container is considered a reusable unit that enhances the functionalities provided by the microservices' containers[4]. At the same time, they can be developed, maintained and deployed individually by the DevOps team. As shown in the figure, we observe a sidecar container deployed against each service, forcing all communication between microservices to pass between their respective sidecars, forming a mesh of nodes ("service mesh architecture"). Such sidecar containers serve as network

[4] `https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/#example-1-sidecar-containers`

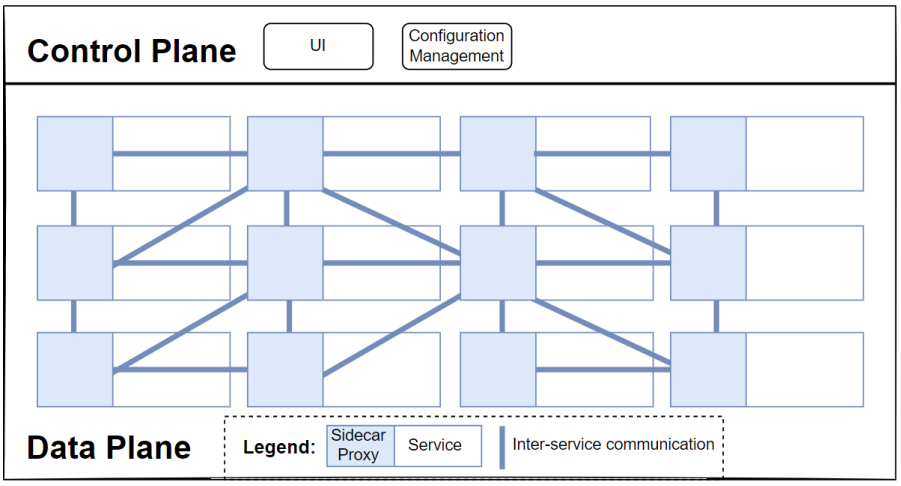

Fig. 2 Overview of a Service Mesh Architecture

proxies, intercepting and further processing communication based on rules set by system admins.

Figure 2 presents a typical service mesh-based microservices architecture consisting of two major components: a control plane and a data plane. According to the architecture definition of Istio[5], the control plane is a set of services that manages and monitors the service mesh proxies according to service mesh configurations provided by DevOps engineers. The data plane accepts the aforementioned configurations and enforces the configurations by forcing communication through the microservices' proxies.

In other words, the service mesh proxies serve as a dedicated mechanism to perform advanced features, including but not limited to monitoring, routing and load balancing. Monitoring often covers the so-called "three pillars of observability data" [34], including (I): "Traces" - the distributed call graph of the monitored services, which helps to understand the topology and detect abnormal behaviour. (II): "Metrics" - the numerical measurement of specific aspects of the health of services. (III): "Logs" - the system logs gathered from distributed services, which record the detailed system events at each time. Routing serves the system to direct the traffic from outside the network to internal services based on user strategies and system logic. On the other hand, load balancers usually follow fixed strategies to distribute workload evenly among identical instances of each service [13].

[5] `https://istio.io/latest/docs/ops/deployment/architecture/`

3 Methodology

3.1 Goal and Research Questions

We rely on the Goal/Question/Metric template [5] to define our study goal as follows: *to analyze* the questions and answers software practitioners posted online; *for the purpose of* revealing the recurrent concerns practitioners face while adopting service mesh in industry settings, and comparing these concerns with those observed in the traditional microservices domain; *with respect to* understanding service-mesh-related discussion topics, adoption goals, symptoms, fixes and the remaining concerns for practitioners and researchers to address; *from the point of view of* service mesh and traditional microservices practitioners and software engineering researchers; *in the context of* common open-source service mesh frameworks and their related questions and knowledge resources on Stack Overflow and dedicated Q&A fora hosted by the open-source frameworks.

We introduce the three research questions (RQs) below to achieve the above-mentioned goal.

RQ1: *What types of questions about service mesh are practitioners asking?* – Throughout the rapid emergence of service mesh frameworks and products, the adoption process has not been trivial for practitioners and companies. Consequentially, many questions regarding the deployment, configuration and operation of a service mesh network exist in today's questions and answers platforms. To the best of our knowledge, such questions generated from practitioners' experiences have yet to be explored, since empirical evidence on service mesh adoption concerns is scarce. We aim to shed light on the main concerns that pose obstacles to adopting the service mesh frameworks.

In RQ1, we focus on understanding the question types and topics in the service mesh domain by gathering and analyzing concerns embedded in question post titles and bodies. We define question types to represent the nature of the inquiry. For example, the "error-related" type represents a practitioner's question on an explicit error or implicit failure while using a service mesh framework. On the other hand, topics are the top keywords expressing the central concern of a group of questions. We derive topics with the help of topic models. We aim to provide an exploratory understanding and definition for each such topic. We also compare these to the top operational concerns of the traditional microservices domain. In the end, we summarize and shed initial insights into the overall state of the service mesh domain by analyzing the question type, acceptance rate and topic definitions.

RQ2: *How did the questions of concern evolve since the emergence of service mesh?* – The earliest service mesh frameworks date to around 2016 and are still under rapid development. To understand how the topics of concern derived from RQ1 have evolved over recent years, we choose to apply a finer-grained topic analysis over time. We do this by analyzing topics across time frames and observing topic persistence, emergence and disappearance. This step allows us to depict critical insights into the characteristics of evolution

and identify potential problems. Ultimately, we aim to find actionable suggestions for focuses and concerns on which to base our future work.

RQ3: *What are the question intentions, adoption goals, error symptoms, and fix patterns of service mesh problems?* – In order to extract insights from the questions and answers practitioners seed while using each service mesh framework, we use qualitative methodologies on the question posts and accepted answers for studied open-source service mesh frameworks. In order to cross-reference our findings with previous RQs, we sample questions for each question type and extract practical insights based on the characteristics of the question types. We extract question intentions for the knowledge-transfer questions to understand what conceptual knowledge practitioners expect to gain. We extract adoption goals for the 352 studied how-to questions. Finally, we analyze the error symptoms from the 475 studied error-related questions and uncover the common fixes to symptoms. We define an error symptom as the explicit failure situation a service mesh framework users face. At the same time, we enumerate possible fixes to such symptoms and define a process of fixing as a common fix if it can cure more than 50% of the particular symptom.

Inspired by previous studies [16,31,54] that systematically derived taxonomies to capture problems in their domain, RQ3 conducts an explorative qualitative analysis in the domain of service mesh networks by building taxonomy trees on sampled questions of each target service mesh framework and evaluate the differences and similarities. We also extract common fix patterns from the qualified answers and construct an error-to-fix mapping table. We include additional manual analysis to explore the embedded knowledge resources practitioners used in the discussions and reveal complexities that require urgent attention in the service mesh domain.

3.2 Service Mesh Frameworks Explored in This Work

To our knowledge, there are more than ten service mesh frameworks in today's market. Since commercial product teams provide comprehensive customer support for their products, users are less likely to ask questions on public platforms. Hence we exclude commercial frameworks during our analysis because of the potential difference in question nature and the shortage of data. Furthermore, we only collect and analyze service mesh frameworks with more than 100 posts combined on Stack Overflow and their own fora. Table 1 depicts the resulting four open-source service mesh frameworks that this study explores, and we briefly introduce each framework.

Envoy: Envoy[6], first developed and open-sourced by Lyft, is designed to provide high-performance process isolation to enhance the transparency of service networks. As Lyft claimed a complete transformation from a monolithic architecture to a service mesh architecture with the help of Envoy in 2017, the project was quickly adopted by service mesh frameworks covered in this

[6] `https://github.com/envoyproxy/envoy`

Table 1 Project Metrics of Open-Source Service Mesh Frameworks

Name	Feature Focus	#Stars	#Open Issues	#Closed Issues	#Past Commits	#Past Releases
Envoy	DP*	20,691	1,223	6,582	12,064	91
Istio	CP/DP*	31,675	573	15,654	18,619	262
Consul	CP/DP*	25,531	991	3,910	18,260	239
Linkerd	CP/DP*	8,960	134	925	1,433	83

*CP: Control Plane, DP: Data Plane

work as a base implementation of their data plane proxies/agents [28]. We do not consider Envoy as a full-service mesh framework because it often only serves as a data plane. Yet, since both Istio and Consul fully support Envoy, and questions usually involve Envoy-related discussions, we include it in our quantitative study of RQ1 and RQ2.

Istio: Istio[7] is an open-source service mesh framework backed by the community. It uses Envoy as its data plane proxy and owns a custom control plane with dashboards. Istio is considered flexible and has gained over 47% market share according to a user survey in 2020[8]. To our knowledge, Istio is the most dominant framework in the market and owns most closed issues, commits and releases over time (see Table 1).

Consul: Consul[9] is an open-source service mesh framework developed by HashiCorp. Unlike Istio, Consul started its evolution in 2014 (version 0.1) as a framework for complex service communications. However, the service mesh components were not formally developed and defined until late 2017. In our study, we discard any data before the framework designer of Consul redefined it as a service mesh framework in 2017. As shown in Table 1, Consul owns 3,910 closed issues with 18,619 commits and 239 historical releases.

Linkerd: Linkerd[10] is an open-source service mesh that graduated from the Cloud Native Computing Foundation and has since gained popularity. However, we only consider and analyze Linkerd for the first two research questions as it has gone through a complete rewrite and framework renaming (the original Linkerd implementation is deprecated). As a result, during the data collection phase, we extracted only 81 questions with accepted answers on Stack Overflow from the Linkerd dataset. Nonetheless, the questions are further divided into two Linkerd implementations (Linkerd and Linkerd2). We could not gather sufficient data for a meaningful manual analysis to generalize practical insights. Thus, we exclude the Linkerd dataset from the qualitative analysis.

Other open-source service mesh frameworks: Our study cannot cover all open-source service mesh frameworks mainly due to the scarcity of public discussion data (fewer than 100 discussion posts after initial filtering described in Section 3.4). Apart from the dominant status of Istio and Con-

[7] `https://github.com/istio/istio`

[8] `https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf`

[9] `https://github.com/hashicorp/consul`

[10] `https://github.com/linkerd/linkerd2`

sul in the industry, both frameworks provide advanced features that other lightweight service meshes do not offer out-of-the-box. Specifically, Istio and Consul offer advanced ability to customize traffic management, security and integration features. On the other hand, Kuma[11] and Traefik Mesh[12] are considered lightweight service mesh solutions, where limited features are offered in comparison to Istio and Consul. For example, secure service-to-service communication is not readily achievable in other frameworks without compromising security constraints.

On the other hand, we aim to focus on directly comparable service mesh frameworks based on similar technologies and architectures. For example, as a newcomer in the open-source world, Cilium mesh[13] relies heavily on eBPF mechanisms that were not involved in other popular service mesh frameworks such as Istio and Consul. Due to its service mesh functionality not appearing until late 2022, we excluded Cilium mesh from the data collection process.

3.3 Data Sources

Previous studies in software engineering [1,3,14,49,50] have used Stack Overflow and/or public fora as a source of mining practitioner concerns and perceptions. We choose to combine Stack Overflow with public fora, since a previous study [45] suggests that particular question types could be discouraged at Stack Overflow, leading some projects to prefer maintaining hybrid support channels, in our case, public Q&A fora.

Table 2 presents a general view of the available data on Stack Overflow and the project-specific fora of the commonly studied [22,24,37,30] open-source service mesh frameworks of Table 1. As mentioned earlier, these frameworks have more than 100 posts combined on Stack Overflow and their own fora. The second column shows the question count on Stack Overflow, and the third column shows the question count on the official Q&A fora of the particular framework. Since practitioners post more questions in the official, project-specific fora than on Stack Overflow, our study uses questions from both types of venues. One complication is that forum discussions lack answer acceptance mechanisms such as voting. Hence in RQ3, we have to manually process the discussion posts based on domain knowledge and the state of discussions within a question thread to determine concluding answers. We exclude the forum data when analyzing answer acceptance in RQ1 due to the aforementioned reason. In summary, Table 2 presents a sufficiently large body of questions applicable to our analysis' data input.

[11] `https://kuma.io/`

[12] `https://traefik.io/traefik-mesh/`

[13] `https://cilium.io/`

Table 2 Count of Relevant Question Posts

Name	# Questions Stack Overflow	# Questions Official Forum	Link to Forum
Istio	2,283	4,054	`discuss.istio.io`
Consul	1,092	1,489	`discuss.hashicorp.com/c/consul`
Envoy	645	N/A	N/A
Linkerd	81	189	`discourse.linkerd.io`

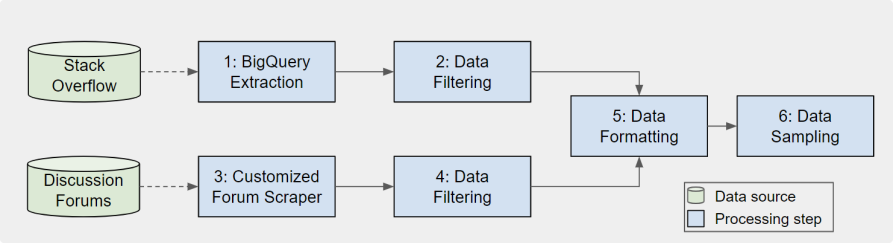

Fig. 3 Data Collection Steps

3.4 Data Collection

Figure 3 depicts the steps we follow to collect data from Stack Overflow, and project-specific discussion fora. [14]:

Step 1: *BigQuery Extraction* – We retrieve the Stack Overflow dataset (4,101 questions) between 2017 and 2022 from the Google Big Query platform[15], which covers the earliest emergence of Service Mesh technology to the recent year. We utilize regular expressions that match the name of open-source service mesh frameworks and closely related terms "service mesh" or "mesh" in question tags, title or body. In order to enable comparison against an established domain, we also collect a separate dataset (7,498 questions) for the broader traditional microservices domain by filtering for the tag "microservices" and excluding the posts related to service mesh, covering data from its emergence in 2015 [47] to 2022.

Step 2: *Data Filtering* – Solely filtering based on regular expression is prone to false positives, especially in cases where irrelevant tags are attached to questions on Stack Overflow [31]. Therefore, we iteratively add conditions to exclude specific keywords from question tags, titles and bodies. For example, we exclude "laravel" since it owns a tool named "Laravel Envoy" unrelated to service mesh.

Step 3: *Customized Forum Scraper* – All public fora of interest use the Discourse open-source forum software[16]. Over the course of several days, we collected public forum data (5,543 posts) starting the creation date of Istio

[14] The scripts to collect, process, filter and analyze data are available in the replication package [15]

[15] `https://cloud.google.com/bigquery`

[16] `https://www.discourse.org/`

and Consul until January 2022, the latest available data hosted on BigQuery for Stack Overflow, avoiding any interruptions of the normal operations of the websites.

Step 4: *Data Filtering* – We first apply a filtering technique similar to the Stack Overflow filtering process of Step 2. Collected data is also filtered based on labels strictly related to service mesh to exclude irrelevant posts regarding other software products covered by the forum. We also manually remove threads that are considered announcements or casual discussions.

Step 5: *Data Formatting* – The raw text goes through common text preprocessing steps, including converting to lower-case, removing non-words, removing stop-words and stemming [4]. To enrich the later topic modelling process with more context, we further generate trigrams, which gather every three consecutive words into a group (e.g., "traffic routing issue") instead of considering only individual terms (e.g., "issue"). In RQ2, the posts are additionally divided based on their creation timestamps into timeframes 3 months apart.

Step 6: *Data Sampling* – As the input of our qualitative study in RQ3, we filter out the forum posts that do not have concluding answers, i.e., we exclude post threads that leave a question open or receive objections from other users. We also remove down-voted questions, since such questions are often considered malformed or lower quality. We discard 758 questions (13.8%) that do not belong to a specific question type by applying a keyword-based classifier introduced in Section 4.1. We sample the remaining question posts based on a 95% confidence level to conduct manual analysis. We split the sample size between the Stack Overflow and official fora data sources based on the ratio of available data after filtering. Table 3 shows the count of questions available for RQ3's analysis after filtering. Since knowledge-transfer questions are scarce after filtering, we evaluate every post instead of focusing on a sample, to provide complete insights.

Table 3 Count of Filtered Question Posts

Framework Name	**# Questions Stack Overflow**	**# Questions Official Forum**	**# Questions Sampled**
Istio (Error)	280	559	264
Consul (Error)	201	257	210
Istio (How-to)	201	351	197
Consul (How-to)	110	164	155
Istio (Knowledge)	18	28	46 (all)
Consul (Knowledge)	7	9	16 (all)

3.5 Analysis Steps

This section introduces the study protocol we follow to conduct our analyses, which is illustrated in Figure 4 (along with the specific output artifacts from each step).

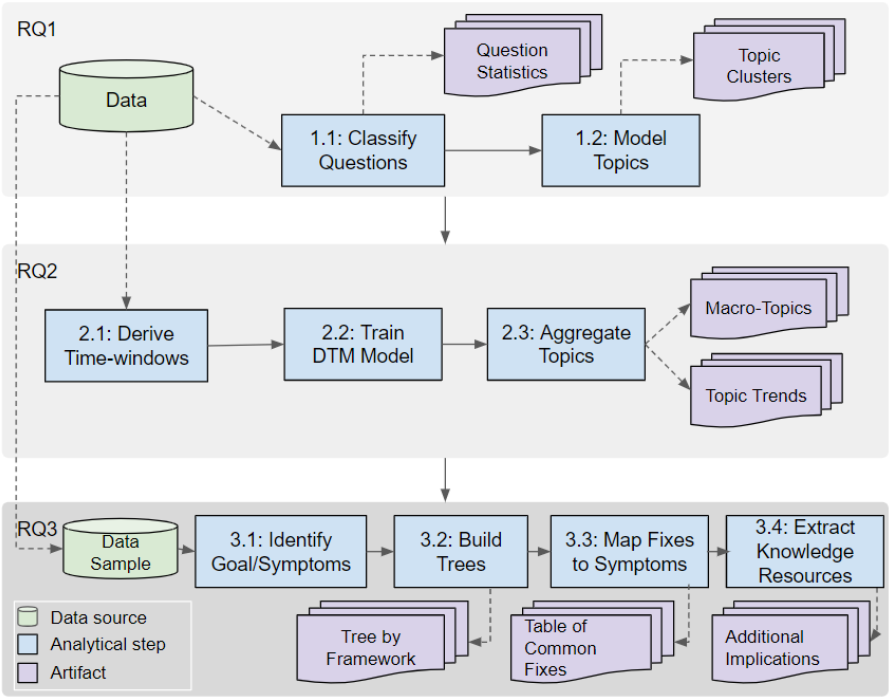

Fig. 4 Study Protocol

Step 1.1: Classify Questions – Before analyzing the question semantics, we first identify the high-level question types via a keyword filtering process based on heuristics, which sorts questions into four types - "How-to," "Error-related," "Knowledge-transfer," and "Misc" questions. The classification is based on conditional filtering for specific keywords that we derive through iterative assessments of the questions in the domain and how practitioners form their questions. For example, we classify a question with both title and body containing "fatal" or "exception" into "Error-related" questions. The full list of keywords is available in the replication package. We only assign one question type to each question in our analysis. Step 1.1 produces the "Question Statistics" artifact, as shown in the figure. We also calculate the acceptance percentage of question posts on Stack Overflow to understand the expertise in the domain.

Step 1.2: Model Topics – With the understanding of differences in question types from the previous step, we extract topics from question posts across all question types combined. We apply an enhanced topic understanding technique called Dynamic Topic Modeling (DTM), which specializes in generating consistent topics across time intervals [8] and has been used by previous studies [3,14,41] to understand the adoption and concerns of new technologies. The choice of DTM is due to our intention to keep topics consistent throughout the study without needing manual topic merging, which could potentially bias our study. For fair comparisons, we apply DTM on the entire set of questions irrespective of time in RQ1 for both the service mesh domain and the traditional microservices domain.

To produce better accuracy and lower the interference from unrelated information, we implemented a threshold to filter out additional words that appear only within less than ten documents or more than 50% of the entire dataset. This step, which essentially expands the stop-word list, helps to eliminate many natural language words that do not present significant meanings.

We tune the hyper-parameters for DTM, aiming for coherent topic clusters. Since the optimal value for hyper-parameters can vary across different datasets, we follow for each domain the so-called "Elbow method" [46]. Through iterative experiments [35], we minimize the marginal returns of the topic coherence **c_v** score, which is empirically shown to be an effective coherence metric for topic modelling.

We use the top keywords produced by the topic modelling process to understand the focus, and label a given topic cluster, which is a common practice used by previous studies [4,26]. In order to best depict the topic in a representative way, we collaborate with a service mesh expert to help labelling the topics based on the mined keywords and analyze representative questions. We give a concrete example for each topic and provide its definition, and provide links to all quoted examples in the Appendix 9. Step 1.2 produces the "Topic Clusters" artifact, as shown in the figure.

Step 2.1: Derive Time-windows – In RQ2, we apply a time-framed analysis to both domains to understand topic evolution and trends. Previous studies take different approaches to understanding the evolution of topics over time. For example, studies commonly [1,26] train individual topic models for each interval and involve manual inferences to depict continuous trends. With the time-aware capability of DTM, we map the topic occurrences to each time interval by retraining the models with additional information on the question creation time. Thus, we conduct trials at different time intervals ranging from 1 month to 12 months. We select 3 months since it best depicts trends over time while suppressing month-to-month fluctuations.

Step 2.2: Train DTM Model – We retrain the DTM models produced in Step 1.2 with additional time intervals derived in Step 2.1. We utilize the DTM model's prediction function to assign a topic to each document and plot their prevalence over time [49]. In our case, the DTM model would provide consistent, framework-agnostic topics across time intervals and thus allow us to depict evolution traits explicitly.

Step 2.3: Aggregate Topics – The application of topic modelling over timestamps yields a trend for each topic over time. In order to depict clear evolution trends considering topic semantics and functionality, we further aggregate similar topics to "macro-topics" according to their definitions from Step 1.2. We sum the aggregated topic prevalence over time and visualize the shifting prevalence of concerns via a moving average trend line. Step 2.3 produces the "Macro-Topics" and the "Topic Trends" artifacts shown in Figure 4.

Step 3.1: Identify Question Intentions, Adoption Goals and Error Symptoms – During the qualitative analysis in RQ3, we start from our understanding of question types in RQ1 and perform detailed analysis to compare and extract user concerns in today's dominant open-source frameworks: Istio and Consul. We conduct our analysis separately on the three major question types (knowledge-transfer, how-to and error-related questions) since they embed different levels of concerns. For knowledge-transfer questions, we focus on understanding the intention to gain certain knowledge in the service mesh domain. For how-to questions, we focus on understanding practitioners' adoption goals, such as customization and performance tuning. For error-related questions, we focus on understanding their symptoms and deriving common fixes.

Since we aim to extract information from a complex domain, we utilize the open card sorting technique [53], which does not rely on pre-determined labels to understand domain knowledge and to identify sampled data. We involve two researchers (a master's student and a postdoctoral researcher), each with over three years of experience in software engineering, to assign macro-topics and to identify intentions, adoption goals, error symptoms and their fixes. We invited an additional domain expert to look at our analysis results to validate and ensure the findings are relevant according to industry experience.

Specifically, we adopt an iterative approach during card sorting to extend our understanding of the service mesh domain and minimize analytical biases by researchers. We start our analysis by using domain-specific knowledge of service meshes and individually assigning labels to each post. Throughout later iterations, we incorporate new understandings of how practitioners form their discussions and refine our knowledge. Finally, we merge and reassign labels when both researchers can achieve a strong agreement to reduce personal biases. Furthermore, if multiple non-related goals/symptoms exist in a question post, we separate them into multiple entries in the final tree. We generalize and combine similar goals/symptoms derived during the first run into fewer and more uniform formats.

Step 3.2: Build Trees – The resulting trees represent and consist of the identified adoption goals and error symptoms from Step 3.1. We assign each goal and symptom node to a parent node denoting a "macro-topic", indicating the main area of concern. We finalize the list of goals/symptoms obtained from the third open card sorting iteration in Step 3.1 to form one tree for each open-source framework to discover the differences and commonalities of goals/symptoms across service mesh frameworks. We visualize each entry with a short descriptive text. Step 3.2 produces the "Tree by Framework" artifact, as shown in the figure.

Step 3.3: Map Fixes to Symptoms – Since practitioners provide useful fixes to errors in the collected dataset, we map the fixes to symptoms by systematically analyzing the question answers. We record the specific approaches to fix each symptom and generalize them into descriptive texts. While some symptoms have diverse fixes, others could have a "common fix" if a particular symptom appears more than once and could be fixed in a common way. We treat a suggested and accepted fix in over 50% of the question posts with the same symptom derived in Step 3.2 as a "common fix pattern" [31]. By the end of Step 3.3, we produce the "Table of Common Fixes" artifact, as shown in the figure.

Step 3.4: Extract Knowledge Resources – Among the posted answers, we identified abundant URLs directing to external documentation websites, blogs and posts. We additionally sample 360 questions out of 5,497 service mesh-related questions (across all question types) from the unfiltered dataset to analyze the knowledge resources practitioners used to answer the questions. We record and examine the content of the sampled knowledge resources and investigate if they serve as direct hints for answering the questions. Based on the results, Section 7 presents additional implications of our study, which hint at future research opportunities.

3.6 Validation of Findings

To validate our findings in the service mesh domain, we contacted an expert with over five years of experience in cloud computing who has contributed significantly to a service mesh startup and the Apache Software Foundation. We provided the expert with the latest draft of our study that included our findings for each research question. We requested the service mesh expert to provide feedback about our findings and possible confounding factors based on their real-world experience. Furthermore, based on the service mesh expert's feedback, we adjusted some of the textual representations of the mined topics (topic names) and of the qualitative analysis results (symptom and fix names) to provide better consistency and clarity.

To avoid bias, we first asked the expert to give insights about our RQ findings without discussing with the paper authors or co-workers. The service mesh expert took time to read our paper and write down the feedback for each major finding. We next conducted an online interview with the expert, where we discussed the practical implications of the service mesh development and adoption process. We exchanged ideas for practical automation and unified implementation standards based on recent developments and trends in related technology.

In this paper, we will refer to the expert as "the service mesh expert." We will discuss the service mesh expert's opinions about our findings in each research question and provide implications backed by practical experience on service mesh technology in Section 7.

4 Topic Analysis on Questions (RQ1 Results)

4.1 Question Types and Acceptance Ratio of Answers

Almost half of the questions in the service mesh domain are error-related, compared to 30% for the traditional microservices domain. On the other hand, conceptual questions are rare in the service mesh domain.

Table 4 summarizes the 4 question types classified by our heuristic-based approach. We conduct a Chi-Squared test on the question type distributions of both domains [32], which yields a low p-value (0.008374) compared to a threshold of 0.05. Therefore we reject the null hypothesis that the two groups have the same distribution of question types, accepting the alternative hypothesis of a significant difference in question type distribution. Within the service mesh domain, the percentage of error-related questions at 48.3% is significantly higher than that of how-to questions (31.6%) and eight times higher than that of knowledge-transfer questions (6.3%).

Table 4 Question Types of Service Mesh and Traditional Microservices

Question Type	Service Mesh	Acceptance Rate	Traditional Microservices	Acceptance Rate
How-to	31.6%	40.4%	36.9%	40.3%
Error-related	48.3%	40.6%	29.9%	37.2%
Knowledge-transfer	6.3%	39.1%	19.8%	43.4%
Misc	13.8%	40.4%	13.4%	41.6%
Total*	**5,497**	**1,297**	**7,168**	**2,559**

*Total accepted answers only cover Stack Overflow data

Surprisingly, only a minority of questions are concerned with general concepts and design choices. We assume that many service mesh practitioners already know basic conceptual knowledge when adopting service mesh, therefore, producing fewer questions on concepts. In comparison, the traditional microservices domain appears more evenly distributed in terms of question types. How-to questions (36.9%) in the traditional microservices domain moderately outnumber error-related questions (29.9%), while knowledge-transfer questions also take up a significant percentage of the total at 19.8%.

Below, we define the four types of questions in the context of service mesh.

How-to questions indicate the user's confusion on the way to solve a problem within the technology involved. Example P01 demonstrates a typical how-to question on integrating a commercial monitoring tool into Istio.

> [P01] *I am new to Istio service mesh. I have to integrate/configure AppDynamics in Istio. I have no clue how to do that. Anything related to this would help. Any example or related links or video...anything.*

Error-related questions either indicate an explicit error or implicit failure in the service mesh. Example P02 demonstrates a typical error-related question on overcoming an error thrown by Istio.

[P02] *Following blog post [URL] I am trying to deploy this sample service on my AWS k8s through Istio gives me "error: no objects passed to apply"*

Knowledge-transfer questions indicate the intention to understand the meaning or existence of a general concept/practice while not digging deep into technical details. Example P03 demonstrates a typical knowledge-transfer question where the user tries to understand why Istio provides an mTLS feature.

[P03] *I try to understand why Istio has the mTLS feature? It enables mutual TLS authentication between all the services in a cluster via automatically issued certificates.*

Misc questions indicate either a combination of previous question types due to a mixture of keywords or questions that do not fit any type. The indications are likely implicit and indirect, meaning our heuristics failed to classify these questions into a specific type. While it is hard to provide a typical example for this question type, any question that does not belong to the above three categories is automatically resolved to this type.

For each question type, we also calculate the percentage of questions on Stack Overflow with accepted answers. This analysis does not consider forum posts because they do not explicitly indicate acceptance status. As shown in Table 4, we arrive at a similar and relatively low acceptance rate of around 40% for both domains, independent from question types, meaning most questions lack a satisfying answer. We presume the service mesh domain in general suffers from a lack of expertise amongst the Q&A platform participants.

Feedback from service mesh expert The service mesh expert mentioned that they are not very surprised that service mesh practitioners ask few questions regarding the conceptual and high-level information of service mesh frameworks, "*Service mesh is a new technology but is derived from and extends the solution to traditional microservices architecture, many of the concepts are similar to those in traditional microservices.*" In terms of the high number of error-related questions and low acceptance rate, the expert pointed out that the increasing prevalence of error-related concerns among service mesh users could be partially attributed to the fact that many cloud vendors build their own products on top of open-source implementations, which often introduce various optimizations, limitations, and tightly integrates with their own commercial products, "*This can cause the examples on the internet to work on one vendor but not work on the other, leading to confusion and frustration for practitioners.*"

4.2 Topics, Macro-Topics and Definitions

We identified 18 topics in the studied service mesh questions across all types, belonging to 5 macro-topics.

Table 5 depicts the topic clusters and top ten keywords that form a particular topic derived from the combined dataset across all question types. We rank the topic clusters in terms of the count of documents categorized into the topic, as obtained via the prediction method of the DTM topic model. With the help of the service mesh expert, we assign one short phrase as the name of each topic based on top keywords and by analyzing typical examples in the dataset.

We manually aggregate the topics into "macro-topics" based on topic definitions (semantics) to provide deeper insights into service mesh adoption concerns. These "macro-topics" reflect a more general area of concerns that are agnostic of framework and technology. The last column of Tables 5 and 6 show the macro-topics that each topic belongs to given their definitions and samples. A topic could be assigned with two macro-topics if the topic consists of questions that relate to multiple concerns.

In the following, we illustrate each topic with a real-life example question from our dataset. We utilize the inference method of the trained dynamic topic model to pick the question samples programmatically by assigning a topic of highest probability for each question post in the dataset. Some question quotes are slightly modified to hide lengthy code snippets and URLs.

Traffic and Gateway: The *Traffic and Gateway* topic comprises issues regarding "ingress" gateways, which describe a component on the edge of a service mesh that handles incoming traffic. Example P04 demonstrates a typical how-to question about a newcomer to the service mesh domain, asking about ingress gateway concepts and basic usages in the context of service mesh.

[P04] *I have recently got into Istio and trying to wrap my head around the gateway concept.*
so fundamentally, I get what it is: an entryway into the service-mesh. However, I do not understand how best to use the gateways.

Routing and Services: The *Routing and Services* topic comprises issues regarding directing and orchestrating service traffic flow through the microservices architecture. As discussed in Section 2, a modern microservices architecture involves many frequently interacting services. Therefore, guiding traffic within the network is essential to serve complete services to external users. Example P05 demonstrates a typical how-to question on routing traffic to an external website.

[P05] *I am trying to set up a Virtual Service such that any traffic on "/" gets routed to google.com. I can get Virtual Services to work with any in-cluster pods/services, but I cannot seem to configure Istio to route to anything outside the cluster.*

Table 5 The Topic Distribution of the Service Mesh Domain

ID	Prev.	Topic	Top Keywords	Macro-Topic
1	9.8%	*Traffic and Gateway*	access, deploy, ingress, namespace, kubernete, yaml, install, apply, ingressgateway, operator	Traffic
2	8.0%	*Routing and Services*	route, traffic, request, virtualservice, rule, host, limit, destination, match, version	Traffic
3	7.8%	*Kubernetes and Deployment*	helm, kubernete, version, install, chart, installation, deploy, setup, enable	Infrastructure
4	7.6%	*Networking and Sidecar*	sidecar, traffic, namespace, proxy, egress, enable, inject, container, injection, kubernete	Infrastructure
5	6.5%	*Machine Management*	client, agent, node, register, address, nod, connect, machine, instance, configuration	Infrastructure
6	6.4%	*Configuration Management*	value, file, template, command, configuration, store, update, config, pass, parameter	Infrastructure
7	6.3%	*Access Control*	policy, authentication, authorization, user, access, jwt, request, deny, rbac, rule	Security
8	6.1%	*Request Monitoring*	request, header, call, trace, dns, grpc, http, com, domain, query	Observability, Traffic
9	5.5%	*Docker and Containers*	container, docker, host, image, machine, compose, swarm, traefik, instance, command	Infrastructure
10	5.2%	*TLS Certificates*	certificate, tls, cert, client, enable, https, configure, manager, sign, ingress	Security
11	5.2%	*Application Deployment*	app, proxy, connect, nginx, log, com, web, frontend, kubernete, container	Application, Infrastructure
12	5.1%	*Spring and Service Discovery*	spring, boot, property, cloud, register, configuration, bootstrap, class, yml, discovery	Application, Traffic
13	4.0%	*Infrastructure Monitoring*	metric, upgrade, plane, control, istiod, version, endpoint, component, configuration, monitor	Observability, Infrastructure
14	3.8%	*Health Check*	health, fail, log, agent, status, leader, info, message, state, watch	Observability
15	3.2%	*Envoy Filters*	envoy, filter, source, connection, debug, proxy, info, pilot, http, tcp	Traffic
16	3.2%	*Node Ports*	port, ingressgateway, tcp, system, default, nodeport, connection, listen, expose, host	Traffic
17	3.1%	*Load Balancing*	load, balancer, balance, release, support, test, ingress, cpu, documentation, configure	Traffic
18	2.5%	*Secret Management*	vault, connect, reset, connection, datum, disconnect, secret, storage, token, failure	Security

Kubernetes and Deployment: The *Kubernetes and Deployment* topic comprises issues regarding installing and deploying the service mesh frameworks in a Kubernetes [12] container environment. Example P06 demonstrates a typical error-related question on installing Istio into a container cluster.

> [P06] *I am trying to install istio using helm. I get an error "forbidden: attempt to grant extra privileges." I am using Azure AKS cluster..*

Networking and Sidecar: The *Networking and Sidecar* topic comprises issues regarding sidecar containers, which involve the proxies in service mesh frameworks. Example P07 demonstrates a typical error-related question on adding a sidecar to an existing Kubernetes pod.

[P07] *I am trying to manually inject istio sidecar into an existing deployment according to the instructions here: [Istio website url] I am getting the following error, [Error trace-back]*

Machine Management: The *Machine Management* topic comprises issues regarding the registration of service nodes (clients) in a service mesh framework and relates to the overall infrastructure. Example P08 demonstrates a typical how-to question on connecting clients to service mesh server nodes.

[P08] *I'm testing a consul server cluster. I am using the go client for this. How do I enter multiple servers for the client to connect to?*

Configuration Management: The *Configuration Management* topic comprises issues regarding the value of specific configuration options to set up a service mesh, for instance, setting up routing rules through YAML manifests. Example P09 demonstrates a typical how-to question on configuring Istio after installation.

[P09] *Every document I found only tells you how to enable/disable a feature while installing a new Istio instance. But I think in a lot of cases, people need to update the Istio configuration. Accessing External Services, in this instance, it says I need to provide "flags-you-used-to-install-Istio", but what if I don't know how the instance was installed? Address auto allocation, in this instance, it doesn't mention a way to update the configuration. Does it imply this feature has to be enabled in a fresh installation?*

Access Control: The *Access Control* topic comprises issues regarding service mesh policy-related features that control the network's security policies. Example P10 demonstrates a typical error-related question on setting up JWT auth for their service backend in Istio.

[P10] *I have implemented an istio policy so that users will need a JWT token to access my backend and admin-backend services. However, it is not letting me through with a valid token. I am running istio-demo on Minikube and have done nothing with my deployment but configure an egress for auth0. Then when I go to apply my policy, I can no longer access these services with my requests.*

Request Monitoring: The *Request Monitoring* topic comprises various issues regarding requests and request header interception. Though this topic is closely related to service traffic management, it also comprises questions on traffic monitoring. Therefore, we assign both macro-topics to this topic. Example P11 demonstrates a typical knowledge-transfer question on the header used for tracing in Istio and Envoy.

[P11] *I was trying to understand the tracing in istio. According to istio documentation, x-request-id can be used for tracing purposes.(https://istio.io/latest/docs/tasks/observability/distributed-tracing/overview/) I see different behaviour in Istio vs pure envoy proxy.*

Docker and Containers: The *Docker and Container* topic comprises issues regarding container orchestration outside of Kubernetes. Example P12 demonstrates a typical question on controlling the startup order of service mesh containers.

[P12] *I am trying to implement a service mesh to a service with Kubernetes using Istio and Envoy. I was able to set up the service and istio-proxy, but I am not able to control the order in which the container and istio-proxy are started.*

TLS Certificates: The *TLS Certificates* topic comprises issues regarding the use of encrypted communication inside a service mesh network. Example P13 demonstrates a typical how-to question on setting up end-to-end TLS in the Consul service mesh.

[P13] *I'm having some difficulty understanding Consul end-to-end TLS. For reference, I'm using Consul in Kubernetes (via the hashicorp/consul Helm chart). Only one data center and Kubernetes cluster - no external parties or concerns.*

Application Deployment: The *Application Deployment* topic comprises issues regarding the applications deployed to a service mesh. Since this topic covers both the application side and infrastructure (Kubernetes deployment), we assign both macro-topics. Example P14 demonstrates a typical error-related question on deploying Istio's BookInfo example application.

[P14] *I am trying to evaluate istio and trying to deploy the bookinfo example app provided with the istio installation. While doing that, I am facing the following issue.*

Spring and Service Discovery: The *Spring and Service Discovery* topic comprises issues regarding Spring-based applications deployed to a service mesh and the discovery of services. To our knowledge, Spring is a dominant microservices framework that frequently integrates with service mesh while providing service discovery functionalities. Since this topic covers both the application side and traffic (service discovery), we assign both macro-topics. Example P15 demonstrates a typical error-related question on setting up a SpringBoot application in Istio.

[P15] *I have an application running in Minikube that works with the ingress-gateway as expected. A spring boot app is called, the view is displayed, and a protected resource is called via a link. The call is forwarded to Keycloak and is authorized via the login mask, and the protected resource is displayed as expected. With Istio, the redirecting fails with the message: "Invalid parameter: redirect-uri."*

Infrastructure Monitoring: The *Infrastructure Monitoring* topic comprises issues regarding service mesh observability that do not concern the observability of actual user applications. Infrastructure monitoring often relies on monitoring metrics such as CPU utilization and memory, which are the performance indicators of a system. This topic includes questions related to the infrastructure of the service mesh frameworks. Therefore, we assign both macro-topics to this topic. Example P16 demonstrates a typical knowledge-transfer question on the possibility of replacing an out-of-box metric in Istio with the desired metric.

[P16] *Is there any way to replace istio-tcp-connections-closed-total metric with istio-requests-total for outbound traffic going through egress gateway?*

Health Check: The *Health Check* topic comprises issues regarding the health-checking mechanisms commonly used to determine if a service is prepared to accept traffic. Example P17 demonstrates a typical error-related question on unexpected behaviour when querying the Consul health-check endpoint.

[P17] *I have a consul agent server on localhost:8500. Then I have a simple HTTP server, which first registers in Consul [snippet], And it answers on getting a request for "healthCheck" [Code Snippet] But Consul's health checker still says that Check is now critical for my service.*

Envoy Filters: The *Envoy Filters* topic comprises Envoy proxies and proxy filtering issues. Example P18 demonstrates a typical error-related question on the connection to a database hosted behind Envoy proxy in Consul.

[P18] *I have a MariaDB database installed directly on a host and a Nomad cluster hosting a phpMyAdmin. Both hosts are inside the same Consul cluster. I'm having some issues trying to connect the phpMyAdmin to my database. phpMyAdmin returns the error:*

Node Ports: The *Node Ports* topic comprises port number-related networking issues in a service mesh regarding inter-communicating components. In the case of Consul and Istio, a node port defines a static port every node in a cluster listens on. Example P19 demonstrates a typical knowledge-transfer question on why Istio opened random ports.

[P19] *Who/what assigns these port numbers? It seems so magical, and I don't like Istio to open up random ports on my nodes; this is a security concern to me!*

Load Balancing: The *Load Balancing* topic comprises issues regarding traffic load distribution in a service mesh architecture. Example P20 demonstrates a typical how-to question on intercepting load balancer traffic using Istio.

[P20] *I want to control/intercept the load balancer traffic using Istio. Istio gives you the ability to add a mixer on a service level, but I want to add some*

code on a higher level just before the request traffic rules get executed. Thus instead of adding actions per service, I want to have some actions executed just after the request was received from the load balancer.

Secret Management: The *Secret Management* topic comprises issues regarding the storage of secrets in service mesh networks since the large number of microservices and the infrastructure itself require credentials to communicate with each other. Example P21 demonstrates a typical error-related question on integrating HashiCorp Vault with Istio.

[P21] *The first issue I had is Vault, and Istio sidecar is not running properly, and the application can not be able to init as below. I tried to use the below annotations to init the first vault, but it did not solve the below issue.*

In order to put the service mesh topics in perspective, Table 6 presents the topics for the traditional microservices architecture, where practitioners dominantly use a hybrid development framework such as SpringCloud[17] to create and deploy applications. While the microservices topic clusters contain a combination of operational and application-side concerns, the latter generally do not apply to service mesh environments because service mesh is intended to solve operational complexities rather than development complexities. As a result, we focus on understanding the prevalence of similar operational concerns (Security, Infrastructure, Traffic and Observability) in the microservices domain and do not discuss in detail the strictly application-related topics (Topic 2,3,4,6,9,10,11,12,13,15,16). We highlight the comparable topics (Topic 1,5,7,8,14) in bold.

Next, we discuss the subset of traditional microservices topics that are closely related to operational concerns, i.e., macro-topics "Traffic," "Security" and "Infrastructure".

Access Control (ID 1): The *Access Control* topic comprises issues regarding the authentication and authorization features within a microservices system. This topic overlaps with the service mesh topic "Access Control". Example P22 demonstrates a typical knowledge-transfer question on the best practices of introducing authentication into microservices.

[P22] *Is it a best practice to have auth as a separate service in micro-service architecture application? I saw in some microservices app, the authentication is part of each micro-services as inbuilt Thanks*

Kubernetes and Deployment (ID 5): The *Kubernetes and Deployment* topic comprises issues regarding the overall containerization problems involving Kubernetes. This topic overlaps with the service mesh topic "Kubernetes and Deployment" due to a common focus on Kubernetes container orchestration and underlying infrastructure. Example P23 demonstrates a typical how-to question on deploying pods in Kubernetes.

[P23] *I use RPC protocol app as a micro-service and an API Gateway in front of them as a proxy. My question is how could I directly access k8s's pod,*

[17] `https://spring.io/projects/spring-cloud`

Table 6 The Topic Distribution of the Traditional Microservices Domain

ID	Prev.	Topic	Top Keywords	Macro-Topic
1	10.2%	**Access Control**	authentication, access, authorization, token, jwt, security, oauth, session, resource, auth	Security
2	10%	Method and Error Handling	method, error, code, response, exception, class, rest, controller, endpoint, value	Application
3	8.3%	Message Queue	communication, consumer, queue, kafka, topic, rabbitmq, connection, consume, publish, bus	Application
4	7.2%	RESTful Design	pattern, design, read, understand, example, implement, resource, book, rest, document	Application
5	6.5%	**Kubernetes and Deployment**	kubernete, cluster, pod, deploy, azure, route, fabric, node, nginx, expose	Infrastructure
6	6%	Spring Framework	module, boot, property, dependency, class, configuration, package, build, repository, error	Application
7	5.9%	**Docker and Container**	docker, container, image, compose, machine, host, error, build, deploy, port	Infrastructure
8	5.9%	**Service Discovery and Load Balancing**	instance, eureka, load, register, discovery, boot, cloud, registry, configuration, balancer	Application, Traffic
9	5.8%	Account Management	customer, store, transaction, account, source, state, command, update, example, system	Application
10	5.7%	Database Transactions	product, table, update, query, information, store, record, data, fetch, item	Application
11	5.5%	Testing and Integration	test, process, job, thread, response, memory, testing, worker, status, integration	Application
12	5.4%	Entity and Domain Logic	entity, model, domain, layer, object, function, business, cache, logic, validation	Application
13	5.1%	Web Development	app, web, frontend, component, page, core, build, backend, react, asp	Application
14	4.5%	**Version and Pipeline**	version, deploy, environment, team, development, deployment, build, production, pipeline, manage	Infrastructure
15	4.2%	Shared Libraries	share, approach, system, code, schema, option, library, dependency, graphql, practice	Application
16	3.6%	Logging and Debugging	log, com, jar, http, error, localhost, port, web, springframework, core	Application

because I must build a connection pool to hold these connections, for example, I have three RPC micro-service pod.

Docker and Containers (ID 7): The *Docker and Containers* topic comprises issues regarding similar concerns with Kubernetes. Unlike service mesh frameworks that rely heavily on Kubernetes, early microservices adopters heavily used Docker to deploy applications, which is also reflected in the "Docker and Container" topic of service mesh. Example P24 demonstrates a typical error-related question on migrating services from localhost to docker swarm.

[P24] *I have spring-cloud app with 3 services: eureka-server + gateway(Zuul) + user-service (2 instances). In localhost, everything is working, and I can access and get a simple string response, but when I deploy the app in docker swarm, I get an exception when accessing this endpoint.*

Service Discovery and Load Balancing (ID 8): The *Service Discovery and Balancing* topic comprises issues regarding traffic-related problems such as service discovery and load balancing strategies. This topic could map to the "Load Balancing" topic in the service mesh domain due to the fact that service mesh frameworks typically rely on existing service discovery solutions that can be used independently. Example P25 demonstrates a typical error-related question on service discovery and load balancers.

[P25] *Now, the issue is that once I give Eureka URL as [URL] in the microservice, it registers only on one instance depending on which Eureka instance the load-balancer has redirected the request. But even though peer-to-peer awareness is setup, microservice is not getting registered on the other eureka instance.*

Version and Pipeline (ID 14): The *Version and Pipeline* topic comprises issues regarding the versioning, isolation and deployment of services so that it does not impact user experience. Although it does not map to a specific topic in the service mesh domain, such capabilities are greatly enhanced when service mesh routing components offer advanced deployment models such as Canary Deployment and Blue-Green Deployment[18]. Example P26 demonstrates a typical knowledge-transfer question on best practices regarding service versioning in a traditional microservices architecture.

[P26] *I go into microservices architecture based on docker, and I have three microservices, which together create one product, for example, "CRM system." Now, I want my client to be able to upgrade his product whenever he wants to. I have 3 different versions of my microservices; which one should the client see? I guess the product version should be independent of microservices because copying one of the microservices versions would make me go into more trouble than having no version at all. So is there any pattern, idea to handle such a situation?*

4.3 Macro-Topic Prevalence and Comparison with Traditional Microservices

Infrastructure-related questions are dominating both in the domain of service mesh and traditional microservices.

The "Traffic" macro-topic in Table 5 taking up to 24.1% topic prevalence is closely related to various aspects of traffic management functionalities involving gateways, routing rules and load balancers [30]. Such high prevalence reflects that traffic management is at the core of service mesh design, since

[18] `https://istio.io/latest/docs/ops/deployment`

rerouting traffic is essentially what a service mesh network is supposed to do. Understandably, users generate many questions regarding such core features.

While the traffic-related macro-topic concerns the core abstractions and functionalities provided by the service mesh technology [44,48], to our surprise, the "Infrastructure" macro-topic has the highest prevalence of 27.4%. This macro-topic is agnostic to specific service mesh frameworks, mainly concerning container orchestration platforms that serve as the underlying infrastructure of service mesh frameworks. Although it is known that service mesh frameworks heavily rely on containerization to provide flexibility and extensibility [19,30], our findings suggest that such dependency on container orchestration platforms involves a much higher degree of complexity than the actual service mesh technology itself. This suggests that service mesh and related emerging technologies would benefit from more research efforts to ease the use of related deployment and containerization technologies as, apparently, they often are problematic for out-of-the-box usage [48].

The "Security" macro-topic relates to access control, TLS certificates and secret management, which covers 15.3% of the total questions, indicating a considerable focus on safeguarding service mesh networks and deployed services. Among the topics concerning security, "Secret Management" related concerns show less prevalence than other topics. In reality, secret storage is often deployed as an external service, such as Hashicorp Vault[19], meaning that part of the questions could be directed to their dedicated platforms without mentioning service mesh, causing an underestimation of the actual importance of this concern.

The "Observability" macro-topic covers questions on monitoring system telemetries as introduced in Section 3.2. Since observability is a software system property that could apply to either application traffic or service mesh internal components, we find that the "Request Monitoring" topic (ID=8) and "Infrastructure Monitoring" topic (ID=14) each relate to the macro-topics of traffic and infrastructure, respectively. We keep both macro-topics in the table to distinguish different focuses of observability in the domain.

The "Application" macro-topic considers the software projects hosted by the service mesh network rather than the service mesh functionalities. The macro-topic adds up to 10.3% of the total questions, indicating a minor concern. Since issues related to application development are not amongst the design goals of the service mesh technology, we do not further investigate this macro-topic in the study.

While service mesh frameworks are deployed as a dedicated layer to solve operations-related complexities, traditional microservices frameworks are intended as a hybrid of development and operations. As suggested in previous discussions, we only compare to the relevant macro-topics, namely "Infrastructure," "Security," and "Traffic".

The "infrastructure" macro-topic covers 22.8% of all questions in the traditional microservices domain, with a focus on containerization technologies

[19] `https://www.vaultproject.io/`

such as Kubernetes and Docker. This macro-topic has significant prevalence compared to other possible aggregations. Infrastructure issues, including containerization and deployment versioning, are presumed to be some of the core operational complexities that concern microservices adoption.

We only observe one security-related topic in the traditional microservices domain, yet "Access Control" ranks as the most prevalent individual topic (10.2%). On the other hand, service mesh security topics are more diverse, including topics on "TLS Certificates" and "Secret Management". Such a difference can be explained by service mesh frameworks' ability to adopt the "zero-trust architecture" pattern, which is a popular emerging security model to counter the surge of cyberattacks and assumes all communication channels between all entities should not be trusted and therefore should enforce encryption [36]. However, implementing a zero-trust architecture requires the ability to intercept the traffic between each service and component, which is only available in the service mesh domain.

Surprisingly, traffic-related concerns are scarce in the microservices domain (5.9%) relative to its application-related questions. To the best of our knowledge, traditional microservices rely on API gateways and load balancers to handle traffic management, yet they do not offer the detailed service-to-service level features brought by sidecar proxies. The maturity around API gateways and load balancers could result in such a low prevalence. Conversely, service mesh traffic management features are more complex and cloud expose more problems as a new technology.

Summary of RQ1: A high percentage at 48.3% of error-related posts in the service mesh domain indicates that most questions are related to unexpected problems. Despite the core concerns around traffic, service mesh still faces major infrastructure concerns involving containerization.

5 Evolution Analysis of Question Topics (RQ2 Results)

5.1 Evolution of Service Mesh Topics

The evolution of service mesh domain topics indicates gradually fewer questions on core (traffic-related) functionalities, while concerns regarding security and infrastructure persist, and observability became an emerging concern.

Figure 5 shows the resulting trends of the traffic-related macro-topic (and its constituent topics). We represent each topic with a solid line with different colours, and the green dots represent their summed values. In addition, we calculate the moving average of summed prevalence values to suppress short term fluctuations in the trends. As a result, we observe in the traffic macro-topic a gradual decrease from 35% to 15% in moving average over the last five years. The impactful topics "Traffic and Gateway" and "Routing and

Services" have significantly shrunk in 2019 and 2020. On the other hand, the "Node Port" and "Load Balancing" topics have persisted steadily over time.

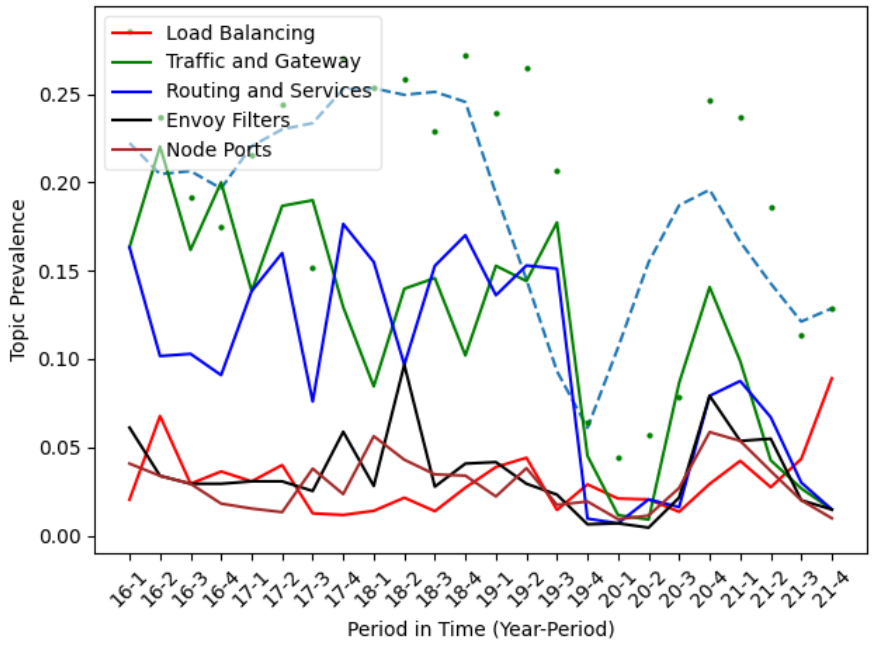

Fig. 5 Evolution of the "Traffic" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

To support our findings, we calculate the mean time to question acceptance from the creation of the question post to question acceptance on Stack Overflow of the topics, resulting in 274 hours for the port-related questions, compared to 353 hours over the entire domain. In our understanding, questions related to ports are more straightforward regarding error messages, thus, easier to fix in a service mesh network. Similarly, load-related questions own a mean acceptance time of 272 hours. Load balancers are often considered a mature technology with well-known strategies [13], possibly contributing to a more limited number of new questions. From Figure 5, we can confirm that, despite a bump in 2020, traffic-related questions diminish over time and pose less concern to practitioners as the service mesh frameworks gradually mature.

Figure 6 reveals the trends within the security-related macro-topic. We observe a spiking outlier of policy-related questions in the second quarter of 2021 (21-2), which seems to coincide with the publication of the first security audit report of the Istio project by the NCC Group[20]. The report reveals several sensitive vulnerabilities regarding Istio and indicates the absence of security-

[20] `https://istio.io/latest/blog/2021/ncc-security-assessment/`

related documentation. In response to the report, Istio issued multiple security patches and published official blogs on enhancing concerns about safety in a service mesh architecture, which could explain the sudden focus of discussions over security policies in this quarter.

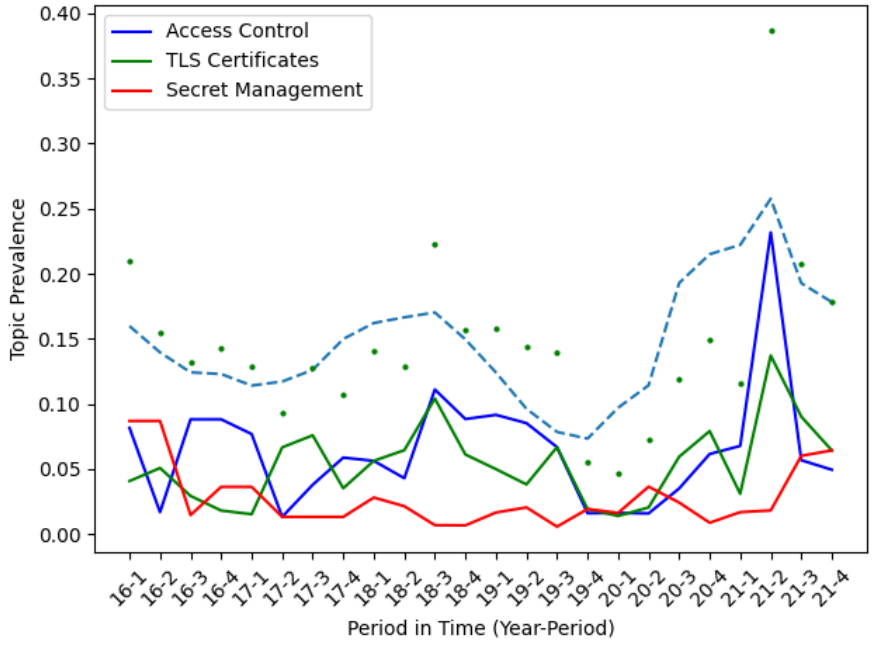

Fig. 6 Evolution of the "Security" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

When considering the trends beyond the outlier, we observe that the security macro-topic started at 10% prevalence overall and has considerably increased over time to surpass the prevalence of the traffic macro-topic. As a result, we can identify security-related concerns as a rising trend that attracts a significant percentage of questions from service mesh practitioners. One previous work [22] points out that the security-related capabilities of service mesh enable a safer network without introducing additional complexities. Yet, another previous work [24] points out that security features in service mesh are fundamentally vulnerable to attacks. Based on the rising trend of security concerns and perceived importance in adoption, we suggest framework maintainers and software engineering researchers pay more attention to service mesh's security concerns to enable practitioners to secure their applications effectively.

Figure 7 reveals the growing trends within the observability-related macro-topic. We want to point out that, as described in Section 4.3, two of the plotted

topics belong to mixed macro-topics, which consists of the observability of service mesh infrastructure (Infrastructure monitoring) and the user applications (Requests monitoring). Overall, we observe a steady and slightly increasing moving average on topic prevalence, matching the growing technical hype on observability solutions[21]. To our knowledge, service mesh-based observability drastically differs from traditional microservices that rely on SDKs and automatic instrumentation agents [29]. Since the service mesh sidecar proxies can only intercept network-level traffic, it is difficult for developers to gain insight into code-level problems at runtime without compromising the non-intrusive nature of sidecars. The difficulty of utilizing such tools in service mesh leads to the requirement for truly non-intrusive ways to monitor system performance, which could be implemented with Extended Berkeley Packet Filter (eBPF), a novel technology to intercept and monitor Linux kernel operations [33].

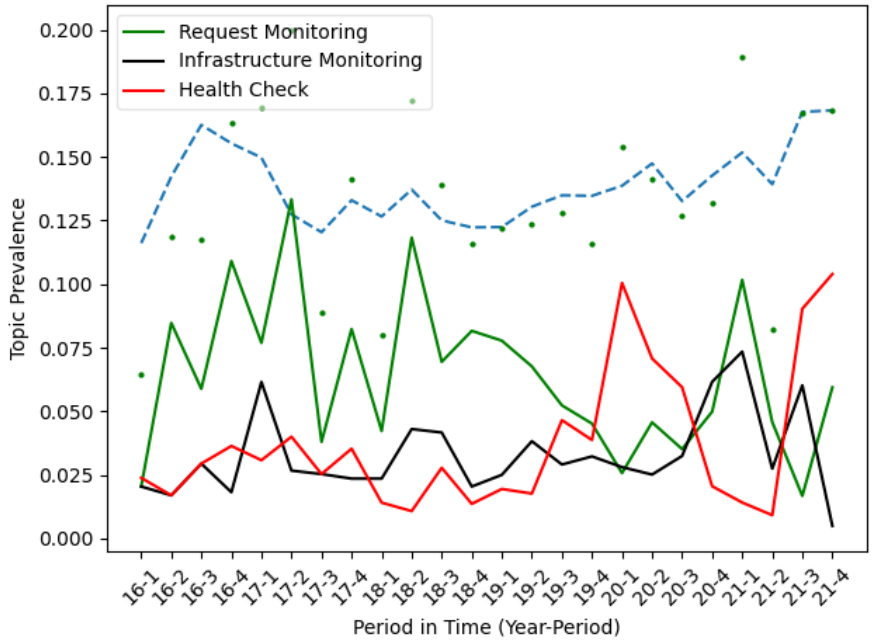

Fig. 7 Evolution of the "Observability" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

Figure 8 reveals the trends within the aggregated infrastructure-related macro-topic. At the early emergence of service mesh frameworks in 2017, practitioners served microservices from hybrid network infrastructures, including virtual machines (VMs), Docker, and Kubernetes [12]. Practitioner preferences

[21] `https://blogs.gartner.com/andrew-lerner/2021/10/11/networking-hype-cycle-2021/`

gradually shifted focus to Kubernetes-like clusters in today's large-scale systems. We observe a gradual decline in the prevalence of infrastructure-related questions until 2018, when service mesh frameworks with underlying containerization platforms rapidly evolved at the earliest adoption stage. Surprisingly,

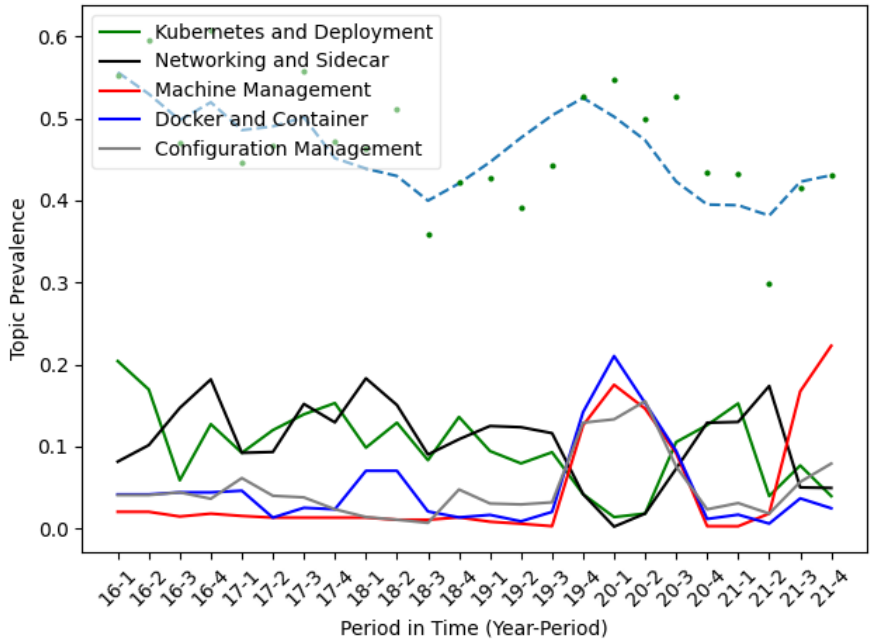

Fig. 8 Evolution of the "Infrastructure" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

with the rapid growth of common service mesh frameworks, the prevalence of infrastructure-related questions rebounded and increased again. The model output shows outliers during 2019 and 2020 as the two topics have swapped prevalence. Despite the outlier period, we observe that the overall prevalence of infrastructure-related questions has increased compared to 2017. The high prevalence of infrastructure-related topics persists even though dominant mesh frameworks release new versions monthly, as shown in Table 1. Next, we compare this trend to the corresponding trend for infrastructure-related questions of traditional service mesh frameworks.

Figure 9 depicts the trend of application-related topics in the service mesh domain. We observe a significant drop of "Spring and Service Discovery" topic prevalence from 20% to less than 5%. We cross-reference this finding with RQ1's analysis of the corresponding service discovery topic in traditional microservices, which represents a minimal prevalence (5.9%). As a result, we

confirm that such topics in both domains, after years of evolution, now cause minimal practitioner concerns.

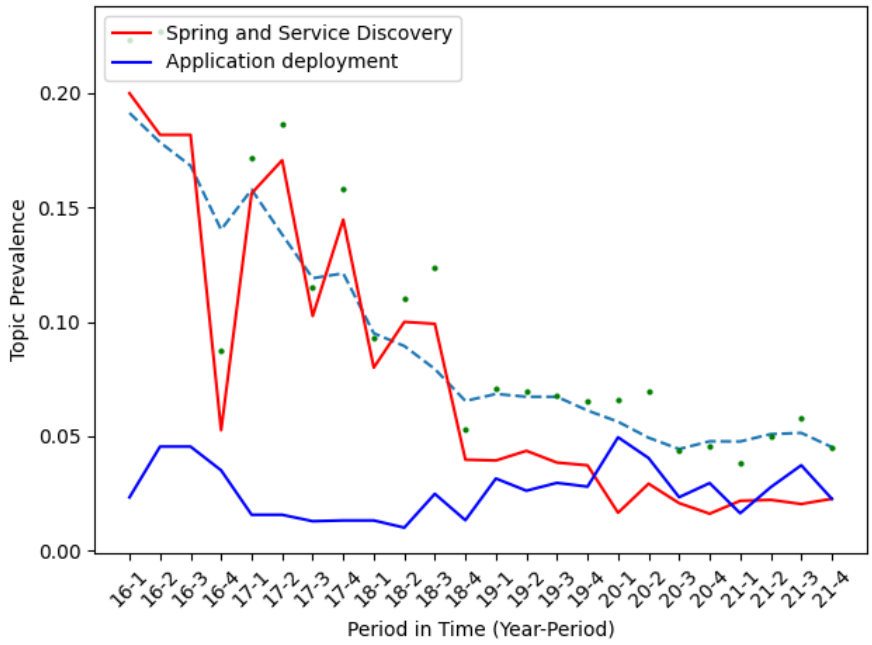

Fig. 9 Evolution of the "Application" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

5.2 Comparison to the Traditional Microservices Infrastructure Domain

The evolution of the infrastructure-related topics in the traditional microservices domain persists, while the service mesh domain sees a minor increase in prevalence over time.

Using a similar time-related analysis as for the service mesh data, and focusing exclusively on the infrastructure-related macro-topic, Figure 10 presents the evolution of the prevalence of aggregated infrastructure topics for traditional microservices. To our surprise, in the relatively mature domain of traditional microservices, where well-known frameworks like Spring Cloud have been dominating for years, infrastructure-related concerns have not reduced in prevalence. Hence, infrastructure-related concerns remain relevant over time for the overall microservices landscape, regardless of the adoption of newer mesh architecture or the evolution of frameworks.

In our topic analysis, keywords including "install," "configuration" and "compose" frequently appear at the top along with those related to container-

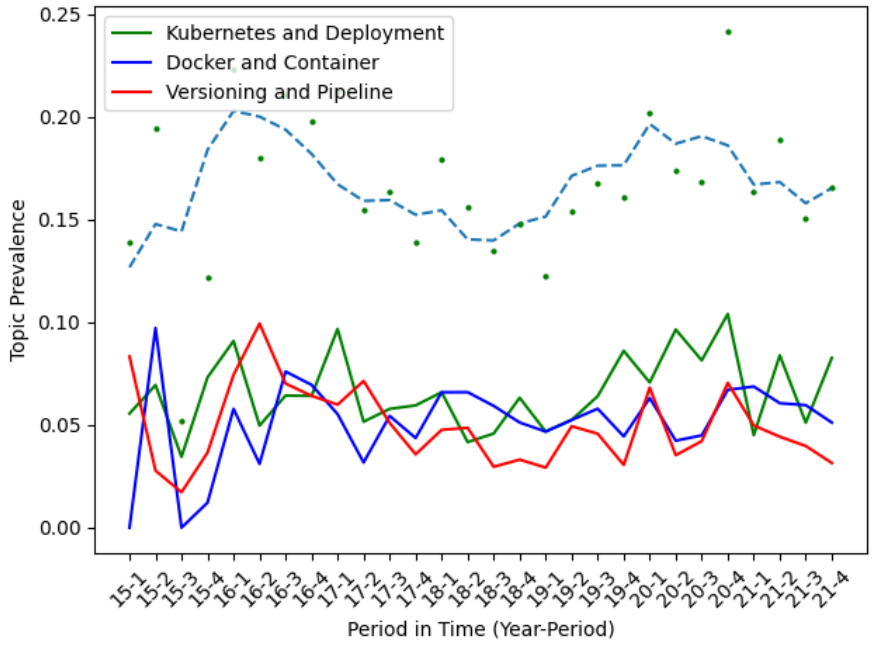

Fig. 10 Evolution of the "Infrastructure" Macro-Topic for Traditional Microservices: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

ization such as "kubernete" and "container" as presented in Table 5. The top keywords suggest most of the infrastructure-related topics relate closely to the usability issues of the installation, update and configuration process in a containerized scenario. We define usability as the easiness of understanding the system setup and correctly configuring the system to work according to user requirements [7]. Given our insights on the evolution of containerization-related topics across time, the persisting infrastructure-related concerns in both domains indicate a pressing need for more empirical understanding to improve the usability of such containerization platforms and ease the adoption of service mesh. To our knowledge, empirical studies on enhancing the usability of modern container orchestration platforms are scarce. An early study on the infrastructure-as-code ecosystem puppet [17] has proposed methods of identifying bad configuration smells. Future research could adopt such methods to modern container orchestration platforms such as Kubernetes, which have quickly evolved into the de facto standard [21].

Feedback from service mesh expert The service mesh expert adds a new perspective on the security trend in the service mesh domain. While supporting our finding that security is an area of increasing importance, the expert indicates the analysis in this research question is likely only the "tip of the iceberg", as many security-related inquiries are likely being conducted through

enterprise support channels and were not discussed in the public domain, even if the practitioners use open-source frameworks. When talking about the specific area to focus on, the expert notes that achieving a zero-trust architecture as explained in RQ1 has become a common goal among service mesh vendors in recent years, and they believe that this direction should draw more research attention as it is challenging to implement and a successful adoption will bring the most out of the service mesh paradigm.

When asked about more insights on the observability domain of service mesh, the expert notes that "*Nearly every cloud vendor has its monitoring/logging product that behaves differently, and some like OpenShift have more strict security policies that can further complicate the adoption and usage of such monitoring technology.*"

Summary of RQ2: Traffic-related topic prevalence fades as service mesh core functionalities mature over time. However, persisting topics in infrastructure indicate remaining concerns that require extra attention. The rise of security and observability concerns indicates future opportunities for novel ideas. Infrastructure-related topics are prevalent in both service mesh and traditional microservices domains.

6 Question Intentions, Adoption Goals, Symptoms and Their Fixes (RQ3 Results)

6.1 Understanding Knowledge Intentions of Istio and Consul from Knowledge-transfer Questions

The intention distribution of the 62 studied knowledge-transfer questions effectively reflects a gap between the carried-over experience from traditional microservices and corresponding service mesh features.

Figure 11 shows the distribution of question intentions among the 62 knowledge-transfer question posts manually labelled based on the question title and body context. Our analysis identified four types of intentions of knowledge-transfer questions:

- **Available functionality**: We identified 36 questions in this category. These questions are intended to understand the features and capabilities of the service mesh frameworks, such as traffic management, security, and telemetry.
- **Best practice**: We identified 11 questions in this category. These questions aim at learning the recommended practices and configurations for using the frameworks effectively and correctly.
- **Architectural decision**: We identified 9 questions in this category. These questions involve the architecture of a service mesh network, such as which

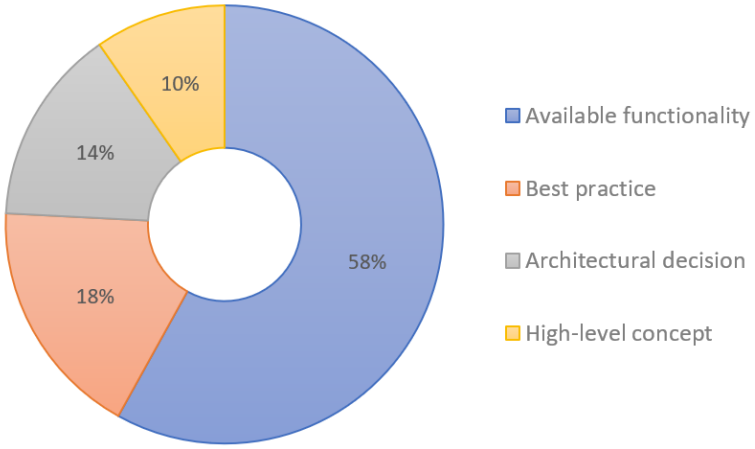

Fig. 11 Question Intentions of Knowledge-transfer questions

components to use and how to structure the mesh to optimize for certain use cases.

– **High-level concept**: We identified 6 questions in this category. Such questions seek to understand the fundamental concepts and principles within the service mesh domain.

We find a dominating 58% occurrence of questions trying to understand the availability of functionalities. In RQ1, we find a low number of knowledge-transfer questions in the service mesh domain in contrast to traditional microservices. During the interview, the service expert indicates that service mesh practitioners benefit from carried-over knowledge from the traditional microservices domain. Nevertheless, the high ratio of inquiries on available functionalities indicates that carried-over knowledge does not necessarily imply a good understanding of service mesh features.

On the other hand, questions regarding best practices and architectural decisions often require a high level of expertise to resolve. Although less dominant than the inquiry for functionality, together they form over 32% of the distribution together. Only 10% of questions ask a high-level concept, confirming our RQ1 findings. In the end, service mesh frameworks own many flexible components, and they can be assembled in many ways in various use cases. Our findings suggest a need for automating the suggestion of the best combination of such components to inform practitioners of available features and their best practices.

6.2 Understanding Adoption Goals of Istio and Consul from How-to Questions

352 studied how-to questions embed 66 unique adoption goals, focusing on integration, customization, and installation. Specifically, Istio practitioners face significant installation and upgrading concerns.

Figures 12 and 13 show the adoption goals for the 197 and 155 analyzed how-to question posts from the open-source Istio and Consul frameworks. Since each service mesh framework involves many adoption goals, here we selectively highlight the frequently co-occurring goals in each macro topic and frequent framework-specific goals. We present the key findings of each macro-topic below:

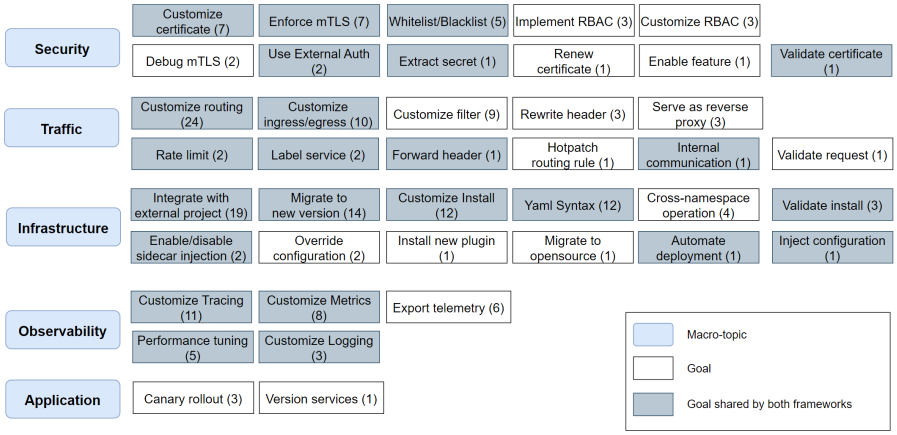

Fig. 12 Adoption Goals of the 197 Studied Istio How-to Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

– Security: The ACL (Access Control List) system is unique to the Consul service mesh; Istio implements the RBAC (Role-based Access Control) system instead. While both systems work towards securing service-to-service communication, we observe a slightly higher (4%) ratio of security-related how-to questions in Istio. When cross-referencing our findings with RQ1, we find that although the prevalence of security-related topics is considerably lower than traffic-related topics, both questions present an equal number of adoption goals.
– Traffic: Interestingly, the traffic topic's adoption goals are highly skewed in Istio, with 24 questions on customized routing based on specific use cases, indicating that practitioner goals cannot be easily achieved with the default settings and configurations provided by Istio. On the other hand, Consul provides out-of-the-box support of "service discovery" features, while Istio relies on external integration to accomplish the goal. As a result, Consul users have the most concerns with this functionality (10 cases).

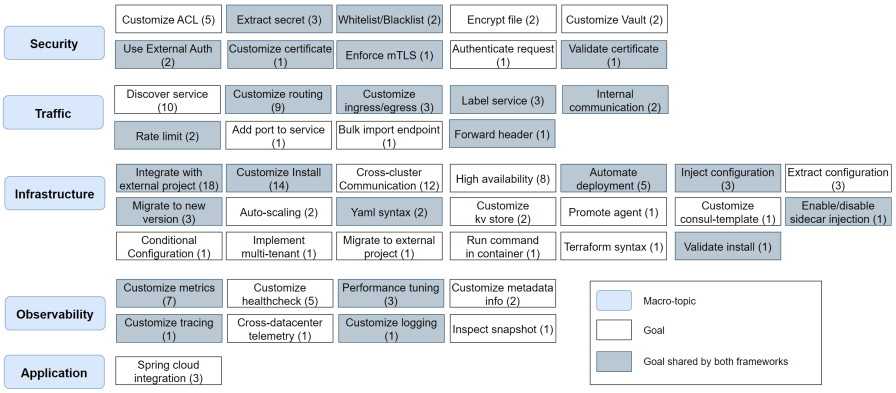

Fig. 13 Adoption Goals of the 155 Studied Consul How-to Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

- Infrastructure (Integration): In both Istio and Consul (19/28 cases), we observe open inquiries regarding the ability to integrate service mesh frameworks with third-party projects such as cloud object storage and legacy applications.
- Infrastructure (Installation): Apart from integration, practitioners of both frameworks raise frequent (12/14 cases) concerns about customization of framework installation spanning various installation methods. Interestingly, only Istio practitioners raise significant number of migration inquiries (14 cases), indicating confusion while attempting to upgrade the Istio version seamlessly. On the side of Consul, we additionally observe significantly more diverse adoption goals (20 of 48 cases) than for Istio (12 out of 41 cases), even though the Consul question samples are smaller, which could indicate more customization needs in the Consul domain.
- Infrastructure (Scaling): The high number of questions related to scaling, such as cross-cluster communication and high availability in Consul (20 cases), suggests that it could be a challenging aspect of using Consul. Surprisingly, cross-cluster questions do not represent a visible distribution in the sampled Istio questions, even though rich multi-cluster functionalities are available according to Istio's documentation.
- Observability: Both Istio and Consul practitioners express customization goals regarding tracing, logging and metrics data. By observing the question content, we find that practitioners generally ask for various telemetry customization that reflects their requirements. In several cases, the users are concerned about the performance impact of collecting telemetry from service mesh, which requires advanced sampling techniques to limit the data size.

6.3 Understanding Error Symptoms of Istio and Consul from Error-related Questions

Construction of symptom trees from the 474 studied error-related posts for Istio and Consul service mesh yields 54 unique symptoms, among them 39 affect both frameworks. On the other hand, both frameworks show frequent unique symptoms that rarely occur in their counterparts.

Figures 14 and 15 show the error symptom trees for the 264 and 210 error-related posts from the open-source Istio and Consul framework. The trees contain error symptoms from the posts that are either explicitly stated as an error or considered misbehaviour of the framework. The left nodes in blue represent the macro-topics whose questions were manually labelled given the actual context of each question and domain knowledge. Each child node represents a generalized symptom directly observed from the question body. Within each symptom tree, we underline symptoms that reappear across multiple macro-topics. To enable a clear comparison between the two service mesh frameworks, we mark symptoms shared by both frameworks in grey, along with the frequency of each symptom.

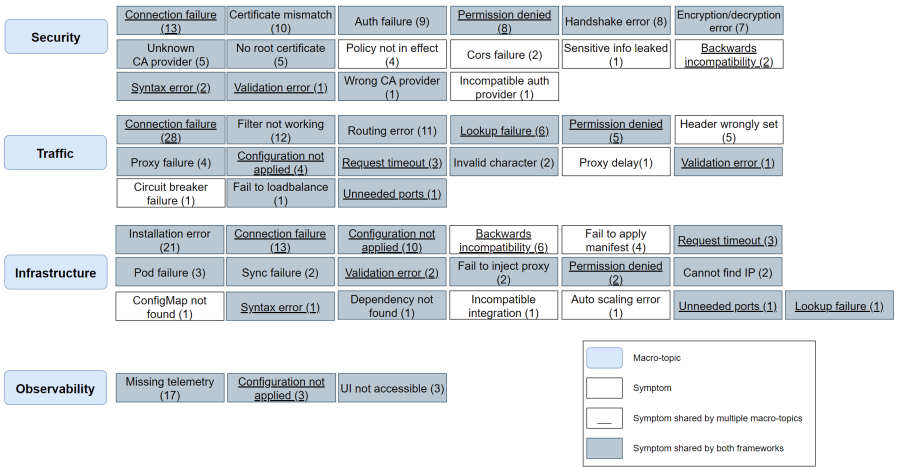

Fig. 14 Error Symptom Tree of the 264 Studied Istio Error-related Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

Within the Istio symptom tree, we generalize 41 error symptoms distributed among four macro-topics of concerns. Similarly, we generalize 43 error symptoms in the Consul case across five macro-topics. Table 7 defines each error symptom. We cross-reference the number of symptoms and their frequencies with RQ1 data to understand whether the most frequent macro-topics found in RQ1 have been significantly discussed in terms of symptoms/fix patterns. In particular, security-related how-to questions account for a lower

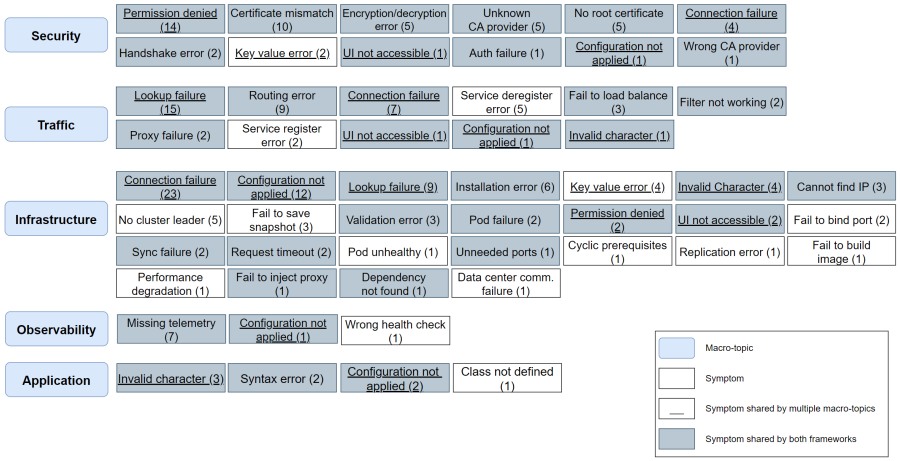

Fig. 15 Error Symptom Tree of the 210 Studied Consul Error-related Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

prevalence in the sampled data (16.7% and 12.9% for Istio and Consul, respectively) and are not comparable with the traffic and infrastructure macro-topic. Yet, security-related error symptoms take significant prevalence (29.9% and 24.3%) in the error symptom analysis. Such imbalance indicates security-related concerns appear more in the form of failures and unexpected behaviour, which is worth further investigation by security researchers.

Specifically, we observe the most appearances of "connection failure", "permission denied," and "Configuration not applied" symptoms. "Connection failure" symptoms account for 54 and 34 cases in Istio and Consul, respectively, meaning the only direct symptom that service mesh practitioners receive in the error log points to a failed network request. "Permission-denied" symptoms occur in 15 and 16 cases in Istio and Consul, respectively, indicating a frequent error on failed attempts to access service mesh network components due to lack of access. Finally, the "Configuration not applied" symptom occurs in 17 cases for both frameworks and dominantly in the infrastructure macro-topic.

Since both frameworks are considered feature-rich and cover common use cases in the microservices domain, the two service mesh frameworks share 39 common symptoms among their union of 54 symptoms across five macro-topics. Most interestingly, the error symptoms also exhibit duplication within different macro-topics of each framework. For instance, the underlined "connection failure" symptoms occur repetitively across three macro-topics in both frameworks. Based on our understanding of the question answers and service mesh frameworks, an erroneous configuration on either traffic routing or underlying deployment infrastructure could cause the same apparent symptom of connection failure. Installation errors, container-related errors and connection failures occur regardless of the framework, supporting our insights from

RQ2 on framework-agnostic infrastructure-related concerns. Previous work on Kubernetes [48] points out that applying default configurations in such distributed systems does not automatically grant error-free environments. Unfortunately, finding suitable configurations in a highly configurable system like service mesh is a non-trivial task [55].

We also observe drastically different dominant error symptoms over macro-topics; we present the key findings of each macro-topic below:

- **Security:** With both frameworks facing diverse error symptoms, Istio practitioners face unique errors that could negatively impact adoption. Although rare (1 case), Istio practitioners report a potential leak of sensitive information by sidecar proxy when intercepting request headers on a default configuration. Such a problem could indicate a potential security flaw. According to the accepted answer, users cannot easily fix the problem by adjusting any configuration, and it is still pending fixing after one year. Additionally, we notice 8 cases where Istio practitioners report backwards-incompatible behaviour when upgrading Istio to a newer version. Specifically, 2 cases break the security-related functionalities, while 6 cases break the proper function of the underlying infrastructure. We do not observe similar error symptoms in the sampled Consul framework posts.
- **Traffic:** Practitioners using the Istio framework dominantly face connection failure symptoms (28), yet Consul practitioners face more failures that relate to service lookup (15), registration (2) and deregistration (5). Istio does not provide comprehensive service discovery and registration features as Consul does. It is possible that such a difference is introduced due to the feature difference between the two frameworks. However, the connection failures representing 32% of Istio traffic-related error symptoms indicate a dominant concern that prevents normal service-to-service communication.
- **Infrastructure (Cross-Cluster):** Practitioners using both frameworks face the most diverse symptoms regarding infrastructure among all five macro-topics. With fewer samples (210 vs 264), Consul practitioners face 25 different infrastructure-related symptoms, which accounts for 58% of all the symptoms in Consul. Apart from connection failures originating from various root causes, we observe unique symptoms such as "No cluster leader", "Replication error, " and "Data center communication failure" related to high availability and cross-cluster communication. When cross-referencing the unique set of cross-cluster adoption goals in Consul, we observe such topics attract major concerns since they cannot be achieved without errors.
- **Infrastructure (Installation):** Surprisingly, while infrastructure-related symptoms are less diverse in Istio, practitioners mostly face installation errors (21 cases) that do not generally occur in the Consul framework (6 cases). Apart from the backwards incompatibility symptoms, Istio practitioners are likely to face errors even in new installations. For example, a user faces an error when installing Istio in WSL2 (Windows Subsystem for Linux):

[P27] *I managed to create a cluster successfully. Then I try to install Istio using helm following the documentation. Everything looks fine till I check the status of Istio pods using for which I get the response [status] The pods continue to stay in ContainerCreating status...*

– **Observability:** While practitioners from both Istio and Consul face errors related to missing telemetry, it is significantly more frequent in the Istio framework (17 cases). Given that Istio provides out-of-the-box distributed tracing for applications while Consul relies on third-party integration, Istio practitioners may face more errors in return for native support for observability.

6.4 Constructing Fix Patterns for Error Symptoms

Less than 50% of the error symptoms have a common fix pattern, and we observe many-to-many error-fix relationships created by unintuitive error messages, causing extra difficulties in root cause understanding.

Table 7 defines each symptom from the above symptom trees within their specific contexts. In the final column of the table, we provide the identifier of a common fix pattern if a given method could resolve over 50% of the questions of the same symptom. Previous work [31] has used 50% as the threshold to define a common fix pattern. We consider a fix as a common fix pattern only if the fixed error symptom appears more than two times and it could be resolved by a particular fix pattern in over 50% of the cases. In other words, only a fix occurring more than once is considered a "common fix pattern".

We briefly explain each fix pattern based on service mesh domain knowledge:

– **Recreate CA**: Although the root cause of an invalid Certificate Authority (CA) can vary, this fix pattern recreates/reloads the certificate authority to ensure it can be properly recognized by both client and server side. This fix pattern can effectively resolve unknown and wrong certificate authority symptoms.
– **Correct certificate signing**: Correcting certificate signing to ensure service mesh components can recognize it. Such fixes can often resolve certificate mismatch errors since the symptom directly hints at the root cause.
– **Check permission policies**: Verifying the access permissions for resources to ensure permissions are not too restrictive. Permission-denied symptoms often hint at restrictive policies.
– **Complete missing policy rules/scopes**: Missing policy rules can impact security measures in the service mesh network, allowing unauthorized access. Adding missing policy rules or flags ensures that security policies are enforced as intended.
– **Check service routing rules**: Correct service discovery/routing to make sure lookup can proceed to resolve service. A wrong destination/missing

Table 7 Symptoms and Their Fix Patterns

Macro-topic	Symptom	Definition	Ref
Security	Permission denied	Permission is not granted for access	S04
	Certificate mismatch	The certificates cannot be used	S08
	Encryption/Decryption error	Failure in end-to-end encryption	
	Unknown CA provider	The certificate authority (CA) is unknown	S01
	No root certificate	No root certificate can be found	
	Connection failure	Access should be granted but denied	
	Handshake error	SSL failure with handshake error	
	Key value error	Failure in key-value secret store	
	UI not accessible	UI related to security not available	
	Auth failure	Authentication failed to work	S03
	Wrong CA provider	The framework indicates wrong CA is used	S01
	Policy not in effect	Security policy not enforced	
	Cors failure	Blocked by cross-origin request policy	
	Sensitive info leaked	Request header leaks sensitive info	
	Syntax error	Configuration syntax not recognized	S07
	Validation error	Fail to validate configuration manifest	S07
	Backwards incompatibility	Security components fail after upgrading	
	Incompatible auth provider	External authentication not working	
Traffic	Connection failure	Connection rejected in traffic routing	
	Filter not working	Envoy filter not working as expected	
	Routing error	Routing is not working at all	
	Lookup failure	Service lookup failure	S05
	Permission denied	Cannot access service due to permission	S04
	Header wrongly set	Wrong manipulation of request header	
	Proxy failure	Proxy container failed	
	Configuration not applied	Config regarding traffic not in effect	
	Request timeout	Request timed out reaching limit	
	Invalid character	Invalid character in RPC communication	
	Proxy delay	Proxy experiencing delays	
	Validation error	Fail to validate configuration manifest	S07
	Circuit breaker failure	Fail to break traffic when triggered	
	Fail to load balance	Load balancer is not working as expected	
	Unneeded ports	Service mesh exposing unnecessary ports	
	Service register error	Service cannot properly register	
	Service deregister error	Service cannot properly deregister	
Infra.	Connection failure	Connection rejected regarding infrastructure	
	Configuration not applied	Configuration provided but not in effect	
	Lookup failure	DNS lookup failure	
	Installation error	Service mesh component installation failure	S06
	Key value error	Failure in key-value secret store	
	Invalid character	Invalid character in RPC communication	
	Cannot find IP	Cannot find Ip address of service	
	No cluster leader	Cannot reach cluster leader	
	Fail to save snapshot	Cannot backup snapshot	
	Validation error	Fail to validate configuration manifest	S07
	Pod failure	Kubernetes Pod failure	
	Permission denied	Cannot access service due to permission	S04
	UI not accessible	UI components not accessible	
	Fail to bind port	Cannot bind to port in system	
	Sync failure	Fail to sync status between nodes	
	Request timeout	HTTP request timeout	
	Pod unhealthy	Container not in healthy status	
	Unneeded ports	Service mesh exposing unnecessary ports	
	Cyclic prerequisites	Cyclic requirements between data centres	
	Replication error	Cross data center replication error	
	Fail to build image	Fail to build container images of component	
	Performance degradation	Component become slower overtime	
	Fail to inject proxy	Fail to inject proxy into the pod	
	Dependency not found	Framework building dependency not found	
	Datacenter comm. failure	Data centre cannot reach each other	
	Backwards incompatibility	Infra components fail after upgrading	
	Fail to apply manifest	Fail to apply configuration manifest	S06
	ConfigMap not found	ConfigMap cannot be found or reached	S09
	Incompatible integration	Integration with third party not working	
	Autoscaling error	Failure in auto-scaling components	
Obs.	Syntax error	Configuration syntax not recognized	S07
	Missing Telemetry	Telemetry is not properly reported	S02
	Configuration not applied	Configuration provided but not in effect	
	Wrong health check	Health check return unexpected result	
App.	UI not accessible	UI related to telemetry not available	
	Invalid character	Invalid character in RPC communication	
	Syntax error	Configuration syntax not recognized	S07
	Configuration not applied	Configuration provided but not in effect	
	Class not defied	Cannot find Java class definition	

Legend:
S01: Recreate CA
S02: Correct monitoring configuration
S03: Complete missing policy rules/scopes
S04: Check permission policies
S05: Check service routing rules
S06: Use proper commands and scripts
S07: Fix configuration syntax
S08: Correct certificate signing
S09: Create ConfigMap

route often leads to various error symptoms, including an apparent connection failure.
- **Use proper commands and scripts**: When practitioners face unexpected results while installing/upgrading service mesh components, correcting the commands/flags in the command line interface (CLI) can usually avoid unexpected results.
- **Fix configuration syntax**: Although configuration validation tools exist for Istio, they cannot verify all scenarios, given their rule-based nature, leading to runtime failures. The fix includes but is not limited to fixing YAML indentation and specific malformed fields. Fixing configuration syntax can avoid validation errors and syntax errors.
- **Correct monitoring configuration**: This error symptom is caused by using default telemetry settings that report incorrect telemetry data and by wrong interpretation of monitoring settings. Fixing the monitoring configuration can often ensure that correct telemetry is reported.
- **Create ConfigMap**: Although this symptom only appears once in Consul, we find this error obvious error symptom exists in service mesh and the overall Kubernetes domain. This problem can always be fixed by recreating the ConfigMap object in the correct place. However, the root cause can vary due to multiple reasons, such as the namespace causing the ConfigMap to be created in the wrong context.

As a result, we could only derive 9 common fix patterns for 12 out of 54 unique symptoms, indicating that most error symptoms, although resolved, do not share a clear connection with their root causes. Additionally, several question authors directly point out their frustrations that framework error messages or current documentation are hard to interpret and do not contribute to direct error resolution. The following quote demonstrates a typical case of user confusion while encountering multiple layers of errors during Consul installation.

[P28] *I can get clients started that don't have docker installed, however on the docker nodes I am unable to get them to start. The first error I got was there are multiple private networks... blah blah... So I edited the config to use the interface name as suggested in discussions. Now I get this error.*

Despite the difficulties in deriving common fix patterns, only a few fix patterns we identified form a one-to-one relationship with an identified error symptom. We observe 29 cases where multiple fix patterns can fix a single error symptom. That is, if the error symptom occurs more than once in our sample, practitioners have provided different ways to fix them. We also find 15 cases where a fix pattern applies to two error symptoms.

These two observations combined form a many-to-many relationship. For example, in the security symptoms of Istio, both "permission denied" and "policy not in effect" can be fixed by "correcting the port mapping naming convention." We can also find symptoms such as "missing telemetry," which can be solved by two fixes. The most extreme example is the "connection

failure" symptom that occurs in multiple places in both symptom trees, which can be fixed in more than 17 ways by observing our limited samples.

Feedback from service mesh expert When asked about our finding on error symptoms, the service mesh expert noted, "*This is really an interesting finding. Service mesh deployments employ the "eventual consistency" paradigm, failing deployments may or may not recover after restarted, depending on the error types, and errors in service mesh frameworks look like Domino Blocks, one error in an upstream node can cause another in a downstream, if the root cause of the errors is not found and fixed, users have to look all errors for possible answers, and if the root cause is found and fixed, all other types of errors disappeared too, this is one of the reasons why the error-fix relationship is not one-to-one exactly.*" With such observations, the service mesh expert emphasized the importance of enhancing error diagnosis mechanisms to help practitioners diagnose and fix complex problems more effectively.

6.5 Analyzing Configuration Changes to Fix Error Symptoms

Finally, we find that in 70% of the cases, minimal changes to the configuration manifest of a specific service mesh functionality could fix the error symptoms.

Service mesh configuration files are written in the YAML format [6], a compact language commonly serving as the orchestration language of modern containerization platforms. Such configuration manifests provide a high-level abstraction of complex operations hidden by service mesh frameworks. Therefore, changing a single line could introduce an impact throughout the service mesh system. We observe that most error symptom fixes (70%) require only minimal changes of fields or lines in the configuration files. In some cases, modifying one configuration line was able to fix a complex error symptom with over 100 lines of error messages. For example, changing a simple port name to "http" of a user's service definition could fix an error symptom of a defective security rule for a question that took 24 follow-up discussions over 13 months before obtaining a viable fix[22].

Previous work [38] has systematically identified the main challenges and activities regarding configuration engineering in practice and provided practical recommendations to address such challenges. However, in the context of service mesh and similar modern containerization technologies, a practitioner could orchestrate and scale hundreds of heterogeneous services to form a large-scale system through simple configurations. Although service mesh and its underlying container orchestration platform have exposed configuration through simple APIs, it also allows the impact of simple configuration changes to propagate beyond the traditional software/hardware environments towards heterogeneous environments across clusters, data centers, and even clouds.

[22] `https://discuss.istio.io/t/jwt-policy-does-not-take-affect/141/25`

Previous work [25] suggests using crowd knowledge to build adaptive configuration suggestions for users of complex software systems. However, in the case of the service mesh domain, users seek the flexibility to customize and extend system capacity to fulfill higher non-functional and functional requirements in today's large-scale systems. Based on our findings in Section 4.1, a crowd-based solution can be challenging to adopt since expertise is likely scarce in the domain. Furthermore, minor modifications in a YAML manifest could lead to significant architectural differences in container topology and software states at runtime[23], producing new challenges beyond applications with predetermined topology. We believe a further understanding of such compact configuration manifests and the adoption of novel ways to recommend suitable configurations that follow best principles [42] (perhaps inspired by those proposed for monolithic systems [56,57]) will be essential for today's popular container orchestration platforms.

Feedback from service mesh expert During the interview, the service mesh expert notes that "*Although many cloud-native engineers jokingly call themselves "YAML engineers", YAML is just a markup language and not a programming language after all. It can define the configuration structures and data types, but it doesn't verify the correctness by itself, leaving the verification to runtime components.*" When asked about the difficulty of coming up with simple configuration changes given the observed symptoms, the service mesh expert adds, "*Sometimes it's even hard to find the corresponding runtime component that consumes a YAML by looking at the symptom and its logs, not to say fixing the error symptoms.*" When talking about the diversity of adoption goals mined, the expert emphasized the need for advanced automation that generates configuration/fixes based on customized scenarios since practitioners of service mesh frameworks tend to aim at a wide range of customization goals. Any valid reference configuration could significantly reduce the risk of errors.

Summary of RQ3: Diverse intentions, goals and symptoms exist in the service mesh domain, exposing concerns of service mesh practitioners around specific topics. We find that service mesh error symptoms do not often have common fixes, since error messages generally do not relate one-to-one to root causes. In many cases, the same error symptom could occur across service mesh macro-topics and receive multiple possible fixes. We also find that container orchestration platforms and configuration engineering play a significant role in error fixing.

[23] `https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/`

7 Study Implications

7.1 For Researchers and Framework Designers

Researchers should investigate service mesh security feature impacts and their adoption complexities: In RQ1 and RQ2, we find that observability and security topics in the service mesh domain attract increasingly more concerns. During the interview with the service mesh expert, we also collected insight from the industry that security concerns could be greatly underestimated due to their nature. The expert adds that zero-trust architectures are prevalent in the service mesh domain amid increasing cyberattacks. Previous work [11] has suggested the superiority of zero-trust architectures. However, the authors suggest that strict security features introduce a non-negligible impact on their users that needs to be explored in academia. In our study, we indeed observe many inquiries regarding the customization of security features of service mesh frameworks. On the other hand, errors frequently occur when practitioners try to apply such features. Future researchers could conduct studies to evaluate the usability of service mesh security-related features, investigate potential usability issues, and understand the source of complexity. This could help identify potential barriers to service mesh adoption and inform the design of more user-friendly security features.

Researchers should investigate novel service mesh observability approaches: The expert also indicates that current service mesh observability features often vary when choosing different vendors and frameworks, leading to adoption complexity and concerns. Given current service mesh features in Istio and Consul, practitioners of both frameworks face observability-related concerns that target application (request monitoring) and service mesh framework themselves (infrastructure monitoring). Practitioners raise a high number of inquiries on customization of telemetry data and face error symptoms in missing telemetry, especially in the Istio framework.

When asked about the observability features, the service mesh expert adds that current service mesh solutions rely on different frameworks that enable tracing, logging, metrics and event telemetry. Since there is no standard to define a healthy service, practitioners are forced to customize the configuration to keep service impact minimal while enabling a complete view of their applications. It is valuable for future observability domain researchers to focus on accurate modelling of system conditions according to industry needs. In this way, practitioners could easily investigate best practices for adjusting observability features given clear requirements.

Researchers should find effective methodologies in configuring complex distributed systems like service mesh: As a highly configurable system that relies on flexible orchestration of software components, service mesh, like machine learning systems [42], comes with high maintenance costs if its configuration problems are not well understood. Service mesh systems rely on compact configuration manifests to define, orchestrate and manage many components. However, when cross-referencing our RQ1 and RQ3 results

on error rates, symptoms and fixes, we find that such high-level abstraction of configurations currently leads to diverse error symptoms that inexperienced practitioners could not quickly fix. Given the service mesh expert feedback in RQ3, configuration validation is hard to achieve and often leads to implicit errors that require expertise currently lacking in the service mesh domain. Our study findings in RQ1 and RQ3 confirm a need for better methodologies and automation to help suggest practical configuration principles linked to practitioners' adoption goals. Future research should focus on understanding the unique challenges that complex distributed systems face in configuration to prevent introducing configuration debts while benefiting from a high-level and compact format.

During the interview with the service mesh expert, we discussed the topic of "What automation is needed and is practical to address the practitioner's diverse concerns given the observation of errors and diverse fixes". Given the recent advances of machine learning techniques, the service mesh expert noted that they believe more work could be done in this area. "*For me, the main culprit of this issue is that errors and fixes are not a one-to-one relationship. And because many users who ask the questions have described their requirements in detail, and the answers have fixes for those errors, I have imagined there were tools that can generate complete and runnable examples for the user requirements, with the errors fixed. With the emerging Large Language Models (LLM), I think these kinds of tools are feasible and would be very helpful for practitioners.*" We conclude the existence of a pressing need for automation that can provide multiple service mesh implementation/configuration options with their advantages, disadvantages, and caveats. Such tool should help practitioners choose the most appropriate service mesh features and usage based on their needs.

Service mesh framework designers should focus on enhancing the framework's error diagnosis capabilities: We identified individual fix patterns that can cure multiple service mesh framework error symptoms due to unintuitive error messages. We also identified various fixes that could cure a single symptom. Therefore, we could not generalize a common fix pattern for most symptoms. We suggest that framework designers systematically identify and better hint at the root causes of error symptoms, eventually enhancing the error messages and eliminating implicit failures (i.e., failures without obvious errors). This way, typical service mesh practitioners could quickly troubleshoot independently or provide precise diagnostic details while asking for help on Q&A platforms. It will also be easier for researchers to generalize common fix patterns and empirically verify best practices that could lead to faster incident recovery.

Another common technique the industry adopts is implementing a unified standard for a range of similar products. For example, OpenTelemetry[24] aims to provide a unified standard for observability data. Nonetheless, when discussing such feasibility in the overall service mesh domain, the service mesh

[24] `https://opentelemetry.io/`

expert points out that unified standards would require collaboration among many experts to exhaust as many usage scenarios as possible. This task is not trivial as service mesh cannot be defined as observability data formats can. When asked what is the best solution to this problem, the expert adds that "*It can be challenging for a single enterprise or organization to gather all the necessary expertise for such implementations, unless they are open-sourced*". The expert believes an open-source standard would benefit practitioners by reducing the need for comparison and decision-making processes. It can also attract more developers to work on the same implementation, ultimately improving service mesh technology's overall adoption and usage.

Service mesh framework designers should practice documentation prioritization: We additionally take a sample of 360 question posts following Step 3.4 in our study protocol of Section 4 to understand more about the available resources that could help with practitioners' concerns. We find 158 external knowledge resources embedded in the question answers, which are essential sources of information to understand the state of the practice regarding specific service mesh frameworks and related technologies. Most interestingly, among 158 external resources, 68 pieces are directly hosted from the official documentation of corresponding service mesh frameworks, meaning that service mesh practitioners who posted the questions could not effectively locate them as helpful information. We speculate that this is due to the highly configurable nature of service mesh systems, resembling a "Too many knobs" scenario [55] that confuses new adopters.

According to our observation of the 158 extracted knowledge resource contents, explicit links between possible symptoms and their fixes (or practices to follow) are scarce, meaning such knowledge is hard to comprehend fully by inexperienced practitioners. Unfortunately, our results on the low answer acceptance rate of the service mesh questions (40%) and a long mean time to acceptance (353 hours) indicate that expertise is generally lacking for the service mesh domain.

Hence, we suggest that service mesh framework designers propose and practice better documentation prioritization techniques to assist new adopters in complex configuration scenarios. For example, one could gather the most frequently asked questions and answers into a FAQ and extract explanations of advanced features into secondary documentation as they can confuse new adopters [24]. We also suggest a review of existing configurable knobs as suggested by previous literature [55] to prevent over-configurable systems.

7.2 For Practitioners Interested in Adopting Service Mesh

New service mesh adopters should take extra caution while exploring advanced capabilities: As the 2021 Gartner report on enterprise networking[25] shows, service mesh has left its "Peak of Inflated Expectations"

[25] `https://blogs.gartner.com/andrew-lerner/2021/10/11/networking-hype-cycle-2021/`

and entered the "Trough of disillusionment" phase, where it could be challenging for early adopters to determine use cases and put service mesh into production environments. Our qualitative analysis suggests that typical practitioners could face complex error symptoms that lead to frustration, which also reflects in the low answer acceptance rate and long mean time to answer acceptance. Practitioners could also face unclear error messages and confusion from available technical resources.

Specifically, based on our analysis in RQ3, Istio adopters tend to face diverse traffic-related and observability-related errors. At the same time, errors related to infrastructure spanning the installation, configuration and connection of service mesh networks occur in both Istio and Consul. Practitioners should familiarize themselves with the error symptoms and common fixes of open-source service mesh frameworks to avoid pitfalls in adoption.

When asked about suggestions to practitioners, the service mesh expert highlights that sometimes a seemingly simple solution to complex problems may involve a higher cost or even lead to building new projects around the solution. Therefore, practitioners should be cautious in adopting advanced features they do not fully understand, as this may lead to technical debt in the long term. The expert also emphasizes the importance of carefully evaluating different implementation options and selecting those that best fit the team's expertise to further reduce the possibility of technical debt.

8 Threats to Validity

Below, we discuss threats to the study validity and the strategies we applied to mitigate these threats, based on literature guidelines [51].

Construct Validity: The heuristic-based classification technique used in Section 4.1 could misclassify some question posts because it is not possible to compose a comprehensive set of filtering keywords. However, the classification process is applied to both the service mesh domain and the traditional microservices to produce an unbiased comparison. Our filtering conditions only used natural language words that do not involve technical details. The potential bias is expected to be minimal and does not contribute to the significant differences in resulting metrics that we discuss in section 4.1.

Another threat is that our topic model is tuned based on the common practices of previous works using an iterative method and evaluation metric [35]. We acknowledge that using other possible hyperparameters could reveal additional information not elaborated in this study. We extensively tested hyperparameters according to the DTM model package documentation and previous literature. It is also possible that using a more advanced clustering model that considers question semantics would yield results that we have yet to discover. However, to our best knowledge, such models require massive data sizes and are hard to implement. We choose to utilize the DTM model as it has been widely applied in recent studies for various topic modelling purposes and is

proposed as an alternative to the original Latent Dirichlet Allocation (LDA) algorithm [8,9].

Internal Validity: One threat to our study is that qualitative analysis can be biased due to its subjective nature. To further enhance our study's validity, we carefully conducted multiple iterations of analysis to refine previously classified questions using newly-learnt domain knowledge. It was not feasible to conduct a full-scale manual analysis of the collected question dataset. As described in the study steps 3.5, we applied sampling with 95% confidence to best preserve its representations. We admit that, by sampling, we could not exhaust every possible fix pattern because of the highly diverse failure symptoms. Yet, the richness of the symptoms and fix patterns that we identified, as well as the fact that towards the end of analyzing our sample no new symptoms/fixes were encountered, provide confidence about the degree of completeness of the qualitative analysis.

Another potential threat to the validity of our study is that we only consulted one service mesh domain expert. While the expert has extensive experience in the domain and has contributed significantly to the community, their opinions and insights may not represent the broader service mesh community. This could lead to biased feedback and limit the generalizability of our study's findings. To mitigate this threat, we conducted an in-depth discussion with the expert. We exchanged our ideas from the perspectives of researchers and practitioners for each implication regarding their practicalness and feasibility. To provide better consistency and clarity, we optimized the textual representation of the topic names and the qualitative labels (without altering the topics' meanings) by incorporating the service mesh expert's feedback. Future research may benefit from soliciting feedback from multiple experts with diverse perspectives.

Conclusion Validity: In the qualitative analysis, we identified the many-to-many relationships between multiple error symptoms and multiple possible fixes. However, it is possible that one can generalize more common fix patterns given a comprehensive knowledge of the framework and eliminate the many-to-many relationship by understanding the root causes of each symptom. Our findings do not necessarily suggest that a common fix pattern cannot be generalized from a given complex error symptom. Still, with the domain knowledge of a typical service mesh practitioner or researcher, they cannot effectively pinpoint a fix pattern in case a root problem can cause multiple errors and lead to confusing error messages.

External Validity: During the RQ3 analysis, we only considered the top two open-source service mesh frameworks because the other frameworks tend to involve minimal discussions. According to market share reports[26] and justifications from Section 3.5, the analyzed frameworks correspond to the top two open-source solutions in the service mesh domain. Due to the nature of commercial service mesh frameworks and paid support services, our findings possibly may not generalize to them.

[26] `https://youtu.be/Du8ImGRd2TI`

Another threat is that we base our study on Stack Overflow and dedicated fora data. We admit that user discussion could happen in alternate sources, such as mail lists and IRC channels. However, such data sources are less structured, making it difficult to determine the "accepted" answer for a given question. Therefore, we choose Stack Overflow and use dedicated fora as supplemental data, since both include sufficient data and are among common practitioner preferences while establishing online support channels [45].

9 Related Work

Table 8 summarizes the main contributions of 11 related papers on service mesh. Next, we briefly go over the relevant publications and focus on an in-depth discussion of the three empirical studies that relate the most to our paper.

A recent study [30] surveyed the emerging service mesh technology and the developing frameworks, including Istio and Linkerd covered in our work. They inspected service mesh capabilities and identified three high-level challenges regarding the quality attributes of a service mesh technology: performance, adaptability, and robustness. The authors suggested the direction of researching service mesh's capabilities regarding edge computation and data analysis. Previous studies in the software engineering domain have covered these aspects, focusing on the adoption of service mesh in edge computation scenarios [18,23,27] and on discovering the automatic operation of service mesh frameworks with machine learning technologies [39,40,43,52]. However, as shown in Table 8, empirical evidence and studies are lacking to prove and pinpoint such challenges and adoption concerns of service mesh. Our work is the first empirical study to systematically elicit and understand real-world concerns faced by service mesh practitioners by analyzing user-seeded posts on Q&A platforms. Next, we cover the three empirical studies most closely related to our study.

Service mesh traffic management: A previous work [37] proposes an empirical study exploring the effects of traffic management-related functionalities in Istio. The authors manually searched and identified a comprehensive set of parameters related to network communication and collected performance metrics in controlled experiments by pinning a single parameter while tuning the remaining. The authors concluded that system administrators should take extra care when configuring the traffic policies because such features could visibly affect each other even with carefully adjusted parameters, leading to degraded performance. The authors also advise against using such mechanisms when traffic overload is not likely to persist since they will inevitably introduce adverse side effects like the connection failure symptoms we discovered in RQ3's Table 7. Similar to our findings, the authors remarked on the difficulty of getting the configuration values right based on actual needs and traffic characteristics, leading to a requirement for concrete suggestions and a reliable source of information.

Table 8 Previous Research on Service Mesh

Publication	Year	Methodology	Contributions
[30]	2019	Survey	Three challenges of service mesh features and future research directions.
[18]	2020	Novel Architecture	A service mesh architecture for Internet of Things (IoT).
[27]	2020	Novel Architecture	A data driven approach to service mesh for IoT.
[23]	2020	Novel Algorithm	A service mesh management algorithm for IoT.
[40]	2020	Novel Algorithm	New algorithms for service coordination.
[39]	2021	Novel Algorithm	Deep learning algorithm for service coordination.
[43]	2021	Novel Algorithm	Adaptive service mesh circuit breaker.
[52]	2021	Novel Algorithm	Adaptive Kubernetes scheduler powered by service mesh.
[24]	2020	Empirical Study (Controlled Experiments)	Assessed the impact of flawed security features in service mesh frameworks and proposed a new threat model.
[22]	2021	Empirical Study (Controlled Experiments)	Identified the challenges to enforce performance constraints in service mesh and the need for reliable benchmark.
[37]	2022	Empirical Study (Controlled Experiments)	Identified effective configurations for Istio circuit breaking mechanisms.
This paper	2023	Empirical Study (Quantitative and Qualitative Case Study)	A systematic study of existing practitioner questions, identifying rising concerns around service mesh security, infrastructure, and observability. Suggesting the need for enhanced methodologies and automation in areas of concern.

Our study's findings on diverse service mesh error symptoms and fix patterns support some claims regarding recurrent configuration challenges. While the general trends of traffic-related work drastically declined over the last two years (see Figure 5), we believe that managing traffic remains a concern as the aggregated category still takes up around 20% of the total topic prevalence. Although previous work [37] derived insights from numerous controlled experiments, an exhaustive search for the best parameters could be unfeasible for practitioners as they cannot deploy their projects into production environments and experiment with the best traffic management policies on a trial-and-error basis. On the other hand, deployment into testing environments requires an accurate understanding of the ever-changing traffic volume and system specification, which is also challenging to provide valuable hints on configuring traffic policies and handling traffic overload. The conclusion

supports our call for empirically verified best practices and indicative error messages whenever an unexpected behaviour occurs in the system.

Service mesh security: Another previous work [24] proposes a controlled experiment focused on service mesh security. The authors indicate that a heavy burden of security overhead was introduced by adopting service mesh frameworks that might offset the benefits of additional workload for system admins. The study tested the service mesh frameworks against adversarial scenarios and discovered design flaws that current service mesh frameworks have. Furthermore, the authors found that even experienced experts could not prevent security issues from happening in over half of the cases. The authors call for future studies to focus on security and amend the security flaws in today's state-of-the-art service mesh frameworks.

Part of our taxonomy regarding security concerns in Table 7 supports the previous work's claims [24] on the pressing need for more attention in this area. Our topic evolution findings within RQ2 (Figure 6) reinforce the authors' conclusions, as we have discovered that the security concern has maintained prevalence over the years. In RQ3, our process to derive the symptoms and fix patterns suggests that such problems in service mesh are hard to pinpoint. Practitioners without deep domain knowledge can only observe the apparent symptoms and understand the root causes from a diverse but scattered selection of knowledge resources (i.e., blogs, books and documentation). Therefore, we support the claim presented by the previous work [24] stating that expert engineers could still face difficulty dealing with mission-critical configurations (i.e., deployment, routing, security) due to inherent software flaws and a lack of reliable best practices.

Service mesh performance: A previous work [22] proposes a study focused on performance (traffic management-related topics) in a context where high-performance networking is essential for microservices to operate normally. Therefore, the researchers conduct controlled experiments to understand computing resource utilization and performance impacts, such as latency, when adopting service mesh in specialized computation domains. The authors concluded that service mesh frameworks face performance challenges when high network performance is one of the top development priorities and call for more action. However, the challenges elicited are derived from the author's experience as a practitioner in the field rather than empirical evidence derived from a systematic study.

Our study derived a list of 18 topics for the service mesh domain. Interestingly, performance was considered an implicit requirement that embeds many of the symptoms that could link to performance problems, such as cross-data-center service mesh, load balancing and proxy delays, which span across three major categories that we aggregated. We realize that service mesh frameworks require more work to optimize when serving fundamental purposes in areas with unique software requirements. As more practitioners adopt the technology, more work should be done to better understand the difficulty of transforming general-purpose service mesh frameworks into specialized solutions for

those areas, given practical suggestions on practices and the correct configurations to use.

10 Conclusions

This work presents an in-depth study of 5,497 service mesh questions on popular question-and-answer platforms, including Stack Overflow and two public fora of open-source service mesh frameworks. While previous works focused on exploring and addressing technical security and traffic-related challenges through controlled experimentation, our study has systematically evaluated existing practitioner question topics, intentions, adoption goals, error symptoms and fixes in the service mesh domain. We provided backgrounds that verify the presence of rising concerns around service mesh security, infrastructure and observability.

We first identified the current state of question types, acceptance and topics via topic modelling. We then explored the evolution of aggregated macro-topics over time and discovered the persisting infrastructure and security-related concerns. We conducted a systematic qualitative analysis, inspecting more than 800 question posts to cross-validate previous research question findings. We compared two popular service mesh frameworks to uncover practitioner knowledge intentions, adoption goals and error symptoms. We reviewed accepted answers and knowledge resources to derive common fix patterns and understand the difficulties in solving the symptoms from a practitioner's perspective. We also uncovered insights on many-to-many error-fix relationships and the impact of minimal configuration manifest changes in highly configurable systems like service mesh.

We suggest researchers to explore the usability and adoption complexities of security-related features to implement secure software architectures. We also identified a need for more user-friendly observability features, which should be modelled accurately according to industry needs. Additionally, we highlight the need for enhanced methodologies and automation to assist practitioners in configuring service mesh infrastructure through configuration engineering. Academia and industry should invest more effort to drive such advances, which could significantly improve the adoption of heterogeneous systems such as service mesh.

11 Conflict of Interest

All authors declare that they have no conflicts of interest.

12 Data Availability Statement

The datasets generated and analysed during the current study are available from the corresponding author on reasonable request.

13 Acknowledgements

We would like to thank the service mesh expert who provided feedback on our findings. Their statements are accounts of personal experience and opinion, and are in no means relate to their current or past affiliations.

References

1. Allamanis, M., Sutton, C.: Why, when, and what: analyzing stack overflow questions by topic, type, and code. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), pp. 53–56 (2013)
2. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 44–51 (2016)
3. Aly, M., Khomh, F., Yacout, S.: What do practitioners discuss about iot and industry 4.0 related technologies? characterization and identification of iot and industry 4.0 categories in stack overflow discussions. Internet of Things **14**, 100364 (2021)
4. Barua, A., Thomas, S.W., Hassan, A.E.: What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering **19**(3), 619–654 (2014)
5. Basili, V., Rombach, H.: The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering **14**(6), 758–773 (1988)
6. Ben-Kiki, O., Evans, C., Ingerson, B.: Yaml ain't markup language (yaml™) version 1.1. Working Draft 2008 **5**, 11 (2009)
7. Bevana, N., Kirakowskib, J., Maissela, J.: What is usability? In: Proceedings of the 4th International Conference on Human-Computer Interaction (HCI International), pp. 1–5 (1991)
8. Blei, D.M., Lafferty, J.D.: Dynamic topic models. In: Proceedings of the 23rd International Conference on Machine Learning (ICML), pp. 113–120 (2006)
9. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. The Journal of Machine Learning Research **3**, 993–1022 (2003)
10. Bourne, V.: Annual apis and integration report - 2022. Tech. rep., Software AG (2022). URL `https://www.softwareag.com/en_corporate/resources/asset/ar/integration-api/apis-integration-microservices-report.html`
11. Buck, C., Olenberger, C., Schweizer, A., Völter, F., Eymann, T.: Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. Computers & Security **110**, 102436 (2021)
12. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J.: Borg, omega, and kubernetes. Communications of the ACM **59**(5), 50–57 (2016)
13. Cardellini, V., Colajanni, M., Yu, P.S.: Dynamic load balancing on web-server systems. IEEE Internet Computing **3**(3), 28–39 (1999)
14. Chakraborty, P., Shahriyar, R., Iqbal, A., Uddin, G.: How do developers discuss and support new programming languages in technical q&a site? an empirical study of go, swift, and rust in stack overflow. Information and Software Technology **137**, 106603 (2021)
15. Chen, Y., Fernandes, E., Adams, B., Hassan, A.E.: Replication package of the paper 'on practitioners' concerns when adopting service mesh frameworks' (2022). URL `https://www.dropbox.com/scl/fo/qrwbg9w9941xoowaoxyb0/h?dl=0&rlkey=lkumrmmc4eez4onu5lnodmz4x`
16. Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., Liu, X.: A comprehensive study on challenges in deploying deep learning based software. In: Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), pp. 750–762 (2020)
17. Cito, J., Schermann, G., Wittern, J.E., Leitner, P., Zumberi, S., Gall, H.C.: An empirical analysis of the docker container ecosystem on github. In: Proceedings of the 14th International Conference on Mining Software Repositories (MSR), pp. 323–333 (2017)

18. De Sanctis, M., Muccini, H., Vaidhyanathan, K.: Data-driven adaptation in microservice-based iot architectures. In: Proceedings of the 3rd International Conference on Software Architecture Companion (ICSA-C), pp. 59–62 (2020)
19. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. Present and Ulterior Software Engineering pp. 195–216 (2017)
20. Esposito, C., Castiglione, A., Choo, K.K.R.: Challenges in delivering software in the cloud as microservices. IEEE Cloud Computing **3**(5), 10–14 (2016)
21. Ferreira, A.P., Sinnott, R.: A performance evaluation of containers running on managed kubernetes services. In: Proceedings of the 11th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 199–208 (2019)
22. Ganguli, M., Ranganath, S., Ravisundar, S., Layek, A., Ilangovan, D., Verplanke, E.: Challenges and opportunities in performance benchmarking of service mesh for the edge. In: Proceedings of the 5th International Conference on Edge Computing (EDGE), pp. 78–85 (2021)
23. Goethals, T., Volckaert, B., De Turck, F.: Adaptive fog service placement for real-time topology changes in kubernetes clusters. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER), pp. 161–170 (2020)
24. Hahn, D.A., Davidson, D., Bardas, A.G.: Mismesh: Security issues and challenges in service meshes. In: Proceedings of the 13th International Conference on Security and Privacy in Communication Systems (SecureComm), pp. 140–151 (2020)
25. Hamidi, S., Andritsos, P., Liaskos, S.: Constructing adaptive configuration dialogs using crowd data. In: Proceedings of the 29th international conference on Automated software engineering (ASE), pp. 485–490 (2014)
26. Hindle, A., Godfrey, M.W., Holt, R.C.: What's hot and what's not: Windowed developer topic analysis. In: Proceedings of the 25th International Conference on Software Maintenance (ICSM), pp. 339–348 (2009)
27. Houmani, Z., Balouek-Thomert, D., Caron, E., Parashar, M.: Enhancing microservices architectures using data-driven service discovery and qos guarantees. In: Proceedings of the 20th International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 290–299 (2020)
28. Klein, M.: Lyft's envoy: Experiences operating a large service mesh (2017). URL `https://www.usenix.org/conference/srecon17americas/program/presentation/klein`
29. Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., Liu, X.: Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering **27**(1), 1–28 (2022)
30. Li, W., Lemieux, Y., Gao, J., Zhao, Z., Han, Y.: Service mesh: Challenges, state of the art, and future research opportunities. In: Proceedings of the 13th International Conference on Service-Oriented System Engineering (SOSE), pp. 122–1225 (2019)
31. Lou, Y., Chen, Z., Cao, Y., Hao, D., Zhang, L.: Understanding build issue resolution in practice: symptoms and fix patterns. In: Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), pp. 617–628 (2020)
32. McHugh, M.L.: The chi-square test of independence. Biochemia Medica **23**(2), 143–149 (2013)
33. Miano, S., Bertrone, M., Risso, F., Tumolo, M., Bernal, M.V.: Creating complex network services with ebpf: Experience and lessons learned. In: Proceedings of the 19th International Conference on High Performance Switching and Routing (HPSR), pp. 1–8 (2018)
34. Picoreti, R., do Carmo, A.P., de Queiroz, F.M., Garcia, A.S., Vassallo, R.F., Simeonidou, D.: Multilevel observability in cloud orchestration. In: Proceedings of the 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 776–784 (2018)
35. Röder, M., Both, A., Hinneburg, A.: Exploring the space of topic coherence measures. In: Proceedings of the 9th International Conference on Web Search and Data Mining (WSDM), pp. 399–408 (2015)

36. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero trust architecture. Tech. rep., National Institute of Standards and Technology (2020)
37. Saleh Sedghpour, M.R., Klein, C., Tordsson, J.: An empirical study of service mesh traffic management policies for microservices. In: Proceedings of the 38th International Conference on Performance Engineering (ICPE), pp. 17–27 (2022)
38. Sayagh, M., Kerzazi, N., Adams, B., Petrillo, F.: Software configuration engineering in practice interviews, survey, and systematic literature review. IEEE Transactions on Software Engineering **46**(6), 646–673 (2018)
39. Schneider, S., Khalili, R., Manzoor, A., Qarawlus, H., Schellenberg, R., Karl, H., Hecker, A.: Self-learning multi-objective service coordination using deep reinforcement learning. IEEE Transactions on Network and Service Management **18**(3), 3829–3842 (2021)
40. Schneider, S., Klenner, L.D., Karl, H.: Every node for itself: Fully distributed service coordination. In: Proceedings of the 16th International Conference on Network and Service Management (CNSM), pp. 1–9 (2020)
41. Scoccia, G.L., Migliarini, P., Autili, M.: Challenges in developing desktop web apps: a study of stack overflow and github. In: Proceedings of the 18th International Conference on Mining Software Repositories (MSR), pp. 271–282 (2021)
42. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., Dennison, D.: Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems **28** (2015)
43. Sedghpour, M.R.S., Klein, C., Tordsson, J.: Service mesh circuit breaker: From panic button to performance management tool. In: Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems (HAOC), pp. 4–10 (2021)
44. Shadija, D., Rezai, M., Hill, R.: Towards an understanding of microservices. In: Proceedings of the 23rd International Conference on Automation and Computing (ICAC), pp. 1–6 (2017)
45. Squire, M.: "Should we move to stack overflow?" measuring the utility of social media for developer support. In: Proceedings of the 37th International Conference on Software Engineering (ICSE), vol. 2, pp. 219–228 (2015)
46. Syed, S., Spruit, M.: Full-text or abstract? examining topic coherence scores using latent dirichlet allocation. In: 2017 IEEE International conference on data science and advanced analytics (DSAA), pp. 165–174. IEEE (2017)
47. Thönes, J.: Microservices. IEEE Software **32**(1), 116–116 (2015)
48. Vayghan, L.A., Saied, M.A., Toeroe, M., Khendek, F.: Deploying microservice based applications with kubernetes: Experiments and lessons learned. In: Proceedings of the 11th International Conference on Cloud Computing (CLOUD), pp. 970–973 (2018)
49. Venkatesh, P.K., Wang, S., Zhang, F., Zou, Y., Hassan, A.E.: What do client developers concern when using web apis? an empirical study on developer forums and stack overflow. In: Proceedings of the 13th International Conference on Web Services (ICWS), pp. 131–138 (2016)
50. Wang, W., Godfrey, M.W.: Detecting api usage obstacles: A study of ios and android developer questions. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), pp. 61–64 (2013)
51. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering, 1st edn. Springer Science & Business Media (2012)
52. Wojciechowski, Ł., Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T., Hong, M.: Netmarks: Network metrics-aware kubernetes scheduler powered by service mesh. In: Proceedings of the 40th Conference on Computer Communications (INFOCOM), pp. 1–9 (2021)
53. Wood, J.R., Wood, L.E.: Card sorting: current practices and beyond. Journal of Usability Studies **4**(1), 1–6 (2008)
54. Wu, M., Zhang, Y., Liu, J., Wang, S., Zhang, Z., Xia, X., Mao, X.: On the way to microservices: Exploring problems and solutions from online q&a community. In: Proceedings of the 29th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 432–443 (2022)
55. Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R.: Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 307–319 (2015)

56. Zhang, S., Ernst, M.D.: Automated diagnosis of software configuration errors. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp. 312–321 (2013)
57. Zhang, S., Ernst, M.D.: Which configuration option should i change? In: Proceedings of the 36th International Conference on Software Engineering (ICSE), pp. 152–163 (2014)

A Index of Q&A Posts

Table 9 Selected Sample Post URLs

ID	URL
P01	https://stackoverflow.com/questions/66872969/how-to-integrate-appdynamics-in-with-istio
P02	https://stackoverflow.com/questions/48567474/istio-error-no-objects-passed-to-apply
P03	https://stackoverflow.com/questions/55453117/what-are-the-benefits-of-enabling-mtls-in-istio
P04	https://stackoverflow.com/questions/62585045/how-to-enable-https-on-istio-ingress-gateway-with-kind-service
P05	https://stackoverflow.com/questions/54659041/route-to-external-site-via-istio-virtual-service
P06	https://stackoverflow.com/questions/53095970/helm-install-istio-returns-forbidden-error
P07	https://stackoverflow.com/questions/57488845/istio-manual-sidecar-injection-gives-an-error
P08	https://stackoverflow.com/questions/52183540/consul-go-client-redundant-server-connection
P09	https://stackoverflow.com/questions/70076326/how-to-update-istio-configuration-after-installation
P10	https://stackoverflow.com/questions/55159292/istio-policy-not-authenticating-jwt
P11	https://stackoverflow.com/questions/67386438/override-x-request-id-header-in-istio
P12	https://stackoverflow.com/questions/52936524/starting-a-container-pod-after-running-the-istio-proxy
P13	https://stackoverflow.com/questions/62023421/hashicorp-consul-how-to-do-verified-tls- from-pods-int-kubernetes-cluster
P14	https://stackoverflow.com/questions/48753297/bookinfo-example-app-crashes-on-istio
P15	https://stackoverflow.com/questions/59251833/spring-boot-minikube-istio-and-keycloak-invalid-parameter-redirect-uri
P16	https://stackoverflow.com/questions/65896941/istio-egress-gateway-use-istio-requests-total-metric
P17	https://stackoverflow.com/questions/62173549/consul-health-checker
P18	https://stackoverflow.com/questions/71479142/consul-connect-envoy-can
P19	https://stackoverflow.com/questions/59303619/how-do-i-install-istio-with-fixed-static-nodeport-assignments
P20	https://stackoverflow.com/questions/53152217/control-intercept-load-balancer-traffic-using-istio
P21	https://stackoverflow.com/questions/65735493/istio-and-hashicorpt-vault-agent-sidecar-not-working-properly
P22	https://stackoverflow.com/questions/69326155/microservice-with-auth-as-separate-service
P23	https://stackoverflow.com/questions/47096064/kubernetes-rpc-micro-service-with-api-gateway
P24	https://stackoverflow.com/questions/52230457/spring-microservices-timeout-docker-swarm
P25	https://stackoverflow.com/questions/59489749/microservices-not-registering-on-all-eureka-instances
P26	https://stackoverflow.com/questions/33202053/product-versioning-microservices
P27	https://stackoverflow.com/questions/66527165/installing-istio-on-wsl2-fails-with-failedmount-for-pods
P28	https://discuss.hashicorp.com/t/consul-client-and-docker-wont-start-due-to-ip-config