

# A QUALITATIVE EVALUATION OF SERVICE MESH-BASED TRAFFIC MANAGEMENT FOR MOBILE EDGE CLOUD \*

CCGRID2022

**Aleksandra Obeso Duque**

Department of Computing Science, Umeå University  
Cloud Systems and Platforms, Ericsson Research  
Stockholm, Sweden  
alekodu@cs.umu.se

**Cristian Klein**

Department of Computing Science  
Umeå University  
Umeå, Sweden  
cklein@cs.umu.se

**Jinhua Feng**

Cloud Systems and Platforms  
Ericsson Research  
Stockholm, Sweden  
jim.feng@ericsson.com

**Xuejun Cai**

Cloud Systems and Platforms  
Ericsson Research  
Stockholm, Sweden  
xuejun.cai@ericsson.com

**Björn Skubic**

Cloud Systems and Platforms  
Ericsson Research  
Stockholm, Sweden  
bjorn.skubic@ericsson.com

**Erik Elmroth**

Department of Computing Science  
Umeå University  
Umeå, Sweden  
elmroth@cs.umu.se

May 13, 2022

## ABSTRACT

Service mesh is getting widely adopted as the cloud-native mechanism for traffic management in microservice-based applications, in particular for generic IT workloads hosted in more centralized cloud environments. Performance-demanding applications continue to drive the decentralization of modern application execution environments, as in the case of mobile edge cloud.

This paper presents a systematic and qualitative analysis of state-of-the-art service mesh to evaluate how suitable its design is for addressing the traffic management needs of performance-demanding application workloads hosted in a mobile edge cloud environment. With this analysis, we argue that today's dependability-centric service mesh design fails at addressing the needs of the different types of emerging mobile edge cloud workloads and motivate further research in the directions of performance-efficient architectures, stronger QoS guarantees and higher complexity abstractions of cloud-native traffic management frameworks.

**Keywords** service mesh · efficient traffic management · mobile edge cloud · multi-access edge computing · performance-demanding applications · software engineering

## 1 Introduction

The advanced digitalization of industry, enterprise, and society drives increasingly strict and stringent application performance requirements compared to what is offered by Centralized Cloud (CC) execution environments. The always

\*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

demanding application performance requirements motivate the distribution and decentralization of computational resources, as in the case of Edge Cloud (EC).

EC allows applications to perform better and, at the same time, reduces network congestion compared to more CC environments. Mobile Edge Cloud (MEC), a.k.a. Multi-access Edge Computing, is a type of EC where the computing nodes are placed geographically close to Mobile Network (MN) nodes, and thus closer to mobile end-users (see Figure 1). Hence, MEC introduces new opportunities to optimize application performance at the cost of an increasing degree of complexity.

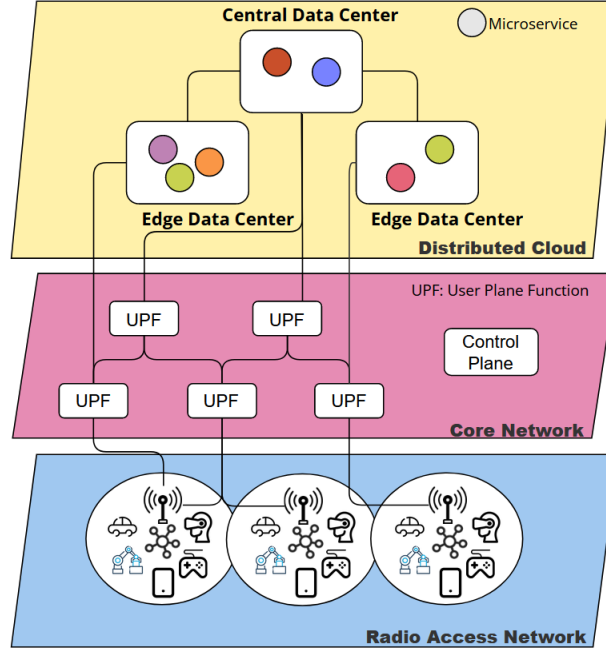


Figure 1: Mobile Edge Cloud architecture. The Mobile Network (MN) is composed by the Radio Access Network (RAN) and the Core Network (CN). The CN is, at the same time, composed by a Control Plane (CP) and a User Plane Function (UPF). On the other hand, the application execution environment is distributed across Centralized Cloud (CC) and Edge Cloud (EC) clusters.

Managing service-to-service communication in a microservice-based architecture is a challenging task. The microservice-based architecture has several advantages, but it also brings complexities related to the development and configuration of disaggregated application components, especially when dealing with a high number of very distributed microservices Fahs and Pierre [2019]. Application-level Service Mesh (SM) is being proposed as a cloud-native approach for traffic management which is getting widely adopted as a de-facto standard. SM supports configurable traffic control, consistent traffic observability, among others. A generic SM is realized as a dedicated infrastructure layer, unifying and centralizing the management and operation of microservice communication (see Figure 2).

However, the design of SM technology is currently oriented towards the needs of generic IT workloads, such as web-based applications, intended to be hosted in CC execution environments. These applications have relatively low performance demands when compared with emerging applications such as the ones targeting automotive, industrial control, Augmented Reality/Virtual Reality (AR/VR) and Internet of Things (IoT) use-cases; meanwhile the latter ones have strong demands in terms of latency, bandwidth and/or reliability. To know if SM can adequately be used for traffic management in MEC, there is a need to analyze in detail their current design.

The research question addressed by this paper is: *How suitable is the service mesh design for traffic management of performance-demanding applications hosted in a mobile edge cloud?* To answer this question, we perform a systematic and qualitative evaluation of state-of-the-art (SoTA) SM design for addressing the traffic management needs of performance-demanding applications running on top of MEC. We do not aim to base our analysis on a comparison across existing SM implementations, but rather on overall SoTA design features of SM.

We start by reviewing related work in this area (see Section 2). Then, we define a set of evaluation criteria based on the characteristics and requirements from MEC and its application workloads (see Section 3). Such evaluation criteria are contrasted with the design drivers, functional and non-functional characteristics of SoTA SM (see Section 4). As a

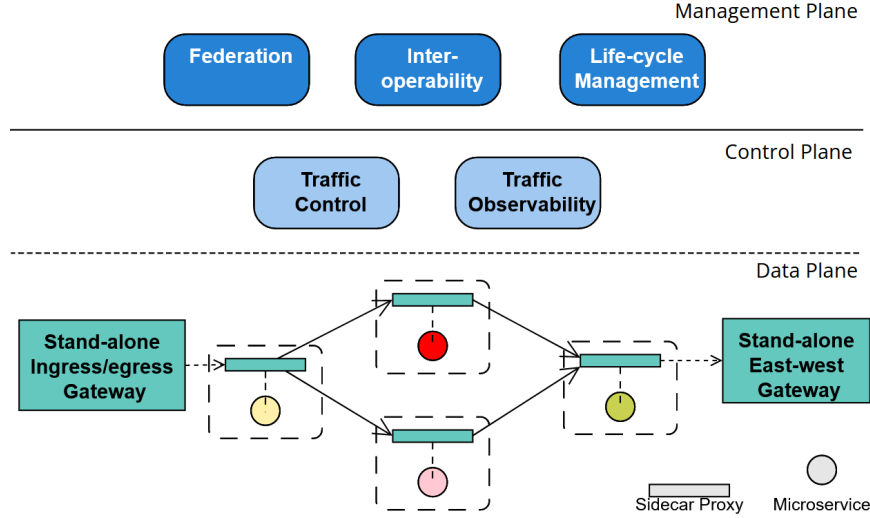


Figure 2: General Service Mesh functionality and architecture. The Service Mesh (SM) is generally split into a Management Plane (MP), a Control Plane (CP) and a Data Plane (DP).

result of this evaluation, we identify SM strengths, limitations and trade-offs that motivate further research in the area (see Section 5).

Our main contributions are two-folded: i. identification of strengths, limitations and tradeoffs associated with SM for MEC, and ii. identification of research challenges and opportunities related to the MEC-specialized SM we envision. With our analysis, we argue why SM fails at addressing the needs of emerging application workloads and foster further research in the areas of performant architecture, differentiated Quality of Service (QoS) assurance, and autonomy of SM. These contributions may represent critical input to standardization efforts related to cloud-native traffic management frameworks not only in the open-source community, i.e., the Service Mesh Interface (SMI)<sup>2</sup>, but also the ones related to 3GPP standards for next-generation mobile networks, in particular, enhanced application architecture for enabling edge applications 3GPP [2021].

## 2 Related Work

Around 2016, one of the first, well-known and well-adopted open-source SM implementations, Linkerd v1, was proposed. Istio and Linkerd v2 started to emerge in 2017 and 2018, respectively. Interesting to note that, at the time of writing, there are more than 20 different implementations of SM<sup>3</sup>. Thus, the SMI initiative is aiming at standardizing the core functionalities of SM. In terms of SM management and operation, tools such as Gloo Mesh and Meshery are starting to appear.

However, most of the SM concepts and implementations are led by the open-source community. Interest from the research community started to emerge in 2017. This section presents related research work in terms of enhanced performant architecture, telco applicability and self-adaptability associated with SM, in relation to our insights discussed in Section 5.

### 2.1 Performant Service Mesh Architecture

In terms of performance enhancement, one proxy-less approach has been proposed. Subhraveti, D., et al. propose AppSwitch, a transport layer network element for modern application architectures Subhraveti et al. [2017]. AppSwitch provides common network connectivity functions such as consistent application identity, application firewall and load balancing logic Cidon et al. [2017] without adding extra cost or complexity. It is implemented as a system call trap/generation function that propagates application location information via a gossip protocol.

<sup>2</sup><https://layer5.io/projects/service-mesh-interface-conformance>, The service mesh interface, accessed 2021-10-04

<sup>3</sup><https://layer5.io/service-mesh-landscape>, The SM landscape: Comparison of service mesh strengths, accessed 2021-10-04

## 2.2 Service Mesh for Telco Domain

Initial efforts are also identified in terms of SM for telco use-cases. For example, Dobaj, J., et al. identify and summarize relevant challenges regarding dependable network design for mission-critical systems, i.e., the need of ensuring dependable system and service connectivity, highly dynamic and flexible connectivity, and holistic and system-wide design approaches to support system interoperability Dobaj et al. [2019]. The authors evaluate three networking patterns: the physical mesh pattern, the logic mesh pattern and the SM pattern for microservice-based applications.

Furthermore, Li, W., et al. explore the challenges, SoTA and future research opportunities of SM Li et al. [2019]. This survey paper revises SM concepts and implementations highlighting opportunities for future research in the area, in particular, the need for further exploration of SM applicability into edge computing environments.

On the other hand, Antichi, G. and Rétvári, G. explore the idea of a full-stack Software-defined Networking (SDN) framework in relation to SM Antichi and Rétvári [2020]. They highlight the need for substantial research in the area, in terms of performant and adaptable architectures and identify high-level SM limitations such as lack of support for carrier-grade performance, QoS assurance and compliance with telco standards and protocols.

## 2.3 Self-adaptability of Service Mesh

In terms of generic self-adaptation support in SM, initial efforts are seen in the research community. For example, Mendonça, N., et al. study the lack of adoption of existing self-adaptation frameworks in the industry community Mendonça et al. [2018]. The paper presents a survey and evaluation of existing solutions in terms of generality vs. reusability and identify multiple self-adaptation patterns, i.e., system-level, infrastructure-level and cross-layer patterns. Furthermore, the authors identify SM as a promising uniform architectural style to handle life cycle of self-adaptive distributed applications.

Saleh Sedghpour, M. R., et al. study and enhance the circuit breaking capability of existing SM Sedghpour et al. [2021]. They argue that existing SM lacks adaptable circuit breaking actuation logic. To tackle this problem, they propose an adaptive controller that avoids overload, mitigates failures, and keeps tail response time below an specified threshold, while keeping throughput at a maximum. To do so, this controller dynamically adjusts circuit breaking queue length thresholds. The proposed adaptive controller can be easily configured to optimize the tradeoff between response time and throughput in a customized way.

Kosińska, J. and Zieliński, K. evaluate how important autonomous performance management is for addressing the highly dynamic requirements characteristic of a cloud-native execution environment Kosińska and Zieliński [2020]. The paper addresses the design and evaluation of a technology-agnostic framework for autonomic management of cloud-native applications, which allows dynamic and on-the-fly reconfiguration. They also compare the proposed framework vs. SM from a design standpoint and identify SM limitations in terms of observability, communication patterns and automated control.

Furthermore, Larsson, L., et al. enhance the resiliency capability of SM-based architectures by adding an adaptive and application-agnostic inter-service response caching mechanism Larsson et al. [2021]. Their proposal is implemented as gRPC interceptors taking care of estimating response longevity and caching. With this enhanced mechanism, they achieve 40% traffic reduction and successful caching for about 80% responses.

On the other hand, Mendonça, N. C., et al. investigate cloud-native self-adaptive SM frameworks by revisiting the history of architectural connectors. These connectors are classified into five different generations, where the fifth one corresponds to the fully-fledged service communication platform, i.e., SM Mendonça and Aderaldo [2021]. The authors argue that none of these generations provide direct support for changing connector's behavior at runtime. They envision software connectors supporting self-adaptation capabilities and allowing operators to customize the logic based on application constraints. They are, at the time of writing, building a prototype of a self-adaptive SM called KubowMesh.

In contrast to the prior knowledge presented in this section, our work focuses on a more thorough evaluation of today's SM design for the specific context of MEC and its different types of workloads, based on its design drivers, and functional and non-functional characteristics. As a result of our work, we identify additional challenges and devise future research directions in this area; especially, for differentiated QoS support as a new feature of a MEC-specialized SM.

### 3 Evaluation Criteria from MEC and its Workloads

In the following subsections, we identify and analyze characteristics and requirements from MEC infrastructure and its different types of application workloads, as the criteria we use in Section 5 for evaluating SoTA SM. The identified characteristics and requirements are used to build profile models for MEC workloads and infrastructure, which are respectively depicted in figures 3 and 4. These diagrams are inspired by SysML requirement diagrams<sup>4</sup>, a general-purpose architecture modeling language for software requirement engineering.

To build the model depicted in Figure 3, we categorize application workloads into three different types of profiles based on their performance requirements: Mission-critical Applications, Bandwidth-demanding Applications and Latency-sensitive Applications. For each of these profiles we identify performance constraints and their associated requirements. Note that we mainly focus on third-party applications that would most likely be provided by non-telco enterprises, i.e., we do not focus on the considerations from cloud-native network functions as such. To build the MEC Infrastructure profile depicted in Figure 4, we identify MEC design constraints, design problems and their implicated requirements.

#### 3.1 Mission-critical Applications

Mission-critical applications perform essential operations for their users and may have extremely high-cost or irreparable losses in case of failure, therefore they have high demands in terms of reliability. In general, users expect to have a high degree of dependability on such applications; thus, there is very low or zero tolerance against application errors or downtime. To deal with such requirements, the applications and the underlying system need to support very high availability, reliability and resiliency/survivability which may be, e.g., achieved based on redundancy and replication strategies (see Figure 3). Examples of such applications are remote operation and control of transport systems.

#### 3.2 Bandwidth-demanding Applications

Bandwidth-demanding applications require the network to transfer high volume of data at very high rates. To achieve those demands, network links need to have enough capacity and advanced mechanisms for traffic management especially when such links are shared with other application loads. An important and complex aspect of this type of applications is the dynamicity of the bandwidth consumption imposed by variable traffic loads and even more, the case of bursty load in which the bandwidth demand is instantaneously spiked up leading to resource starvation (see Figure 3). Examples of such applications are data collection and processing from IoT sensors.

#### 3.3 Latency-sensitive Applications

Latency-sensitive applications require very low delays since they have very low time frames in which responses are expected. In general, they require high application responsiveness. The end-to-end time constraint can be defined in units of milliseconds or even hundreds of microseconds. To deliver such high responsiveness, these applications need to be supported by real-time software and networks. A key aspect to be supported is reliable/bounded latency; in such case it is important to reduce or eliminate jitter (see Figure 3). Examples of such applications are cloud gaming, AR/VR and industrial remote control.

#### 3.4 Mobile Edge Cloud Infrastructure

Highly distributed cloud offers several advantages in terms of e.g., augmented overall computational capacity for large-scale application deployments and geographical redundancy for diverse failover scenarios. When compared to CC, the individual EC sites can also be considered more resource-constrained. Furthermore, EC presents higher resource and performance heterogeneity.

MEC nodes are located in proximity with access network nodes, and thus end-users; this translates into lower transport delays and less network congestion. Distributed cloud allows applications to perform better. MEC can host performance-demanding workloads from third-party application developers or content providers, but can also host Cloud-native Network Functions (CNF) from network providers/operators. To provide optimized QoS, the MN implements differentiated QoS assurance mechanisms based on, e.g., traffic flows, network slicing and Service Function Chaining (SFC) for optimized traffic steering.

However, to provide enhanced end-to-end performance guarantees, it is not enough to merely deploy EC data centers geographically closer to the MN's points of presence. It is also required to have tight integration, awareness, alignment

<sup>4</sup><https://sysml.org/sysml-faq/what-is-requirement-diagram.html>, What is a SysML Requirement diagram?, accessed 2021-10-12

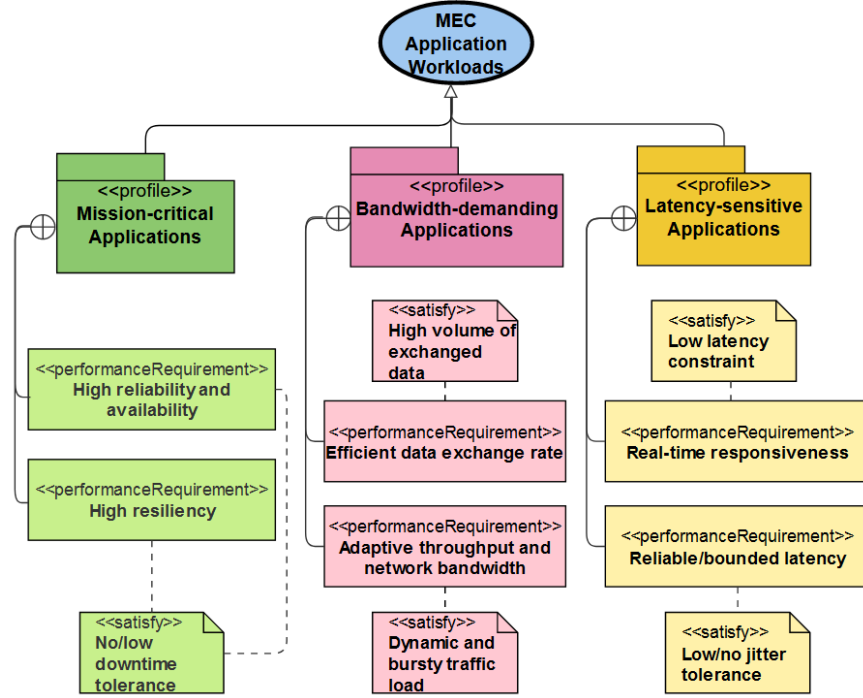


Figure 3: Evaluation Criteria: Characteristics and requirements of different types of performance-demanding applications.

and coordination between both technology domains, the MN and the EC. Performance-demanding applications impose the need for more holistic traffic management approaches, with federation and interoperability considerations.

On the other hand, MN traffic can be considered highly diverse and highly dynamic, not only in terms of the traffic load itself but also due to the need of providing mobility support. Mobile end-users expect to have seamless, uninterrupted, non-degraded Quality of Experience (QoE), even upon mobility events. Figure 4 summarizes MEC’s characteristics and requirements.

## 4 Service Mesh Characteristics and Design Drivers

In this section, we identify functional and non-functional characteristics of SoTA SM and its associated design drivers which we use to build the block diagrams depicted in figures 5 and 6. In Section 5, these design characteristics are evaluated based on the criteria identified in Section 3.

To build the functional diagram depicted in Figure 5, we categorize collected design features into conditions/options of control, actuation mechanisms and observability mechanisms. In the case of the diagram depicted in Figure 6, we identify non-functional characteristics and for each of them we specify the mean or mechanism on which such characteristic relies on. We also identify design drivers and their associated use-cases.

Note that we do not perform a comparison across existing SM implementations; we rather collect, to the best of our knowledge, the full set of features of SoTA SM design. To a great extent, our analysis is based on features from Istio since it is one of the most well-adopted, mature and feature-full implementations of SM, in particular due to its support for multi-cluster, multi-network deployment models needed in the MEC context. However, we also consider features from other well-adopted SM implementations such as Linkerd, Consul and Kuma plus additional features that are not currently supported by such implementations, i.e., the ones proposed by the research community.

### 4.1 Functional Characteristics

In terms of traffic management, SM provides a set of functional features related to traffic control and observability. For actuation mechanisms, SM supports functionalities such as traffic routing, traffic mirroring, traffic splitting, load balancing, rate limiting, circuit breaking, retries and inter-service response caching (see Figure 5). For observability

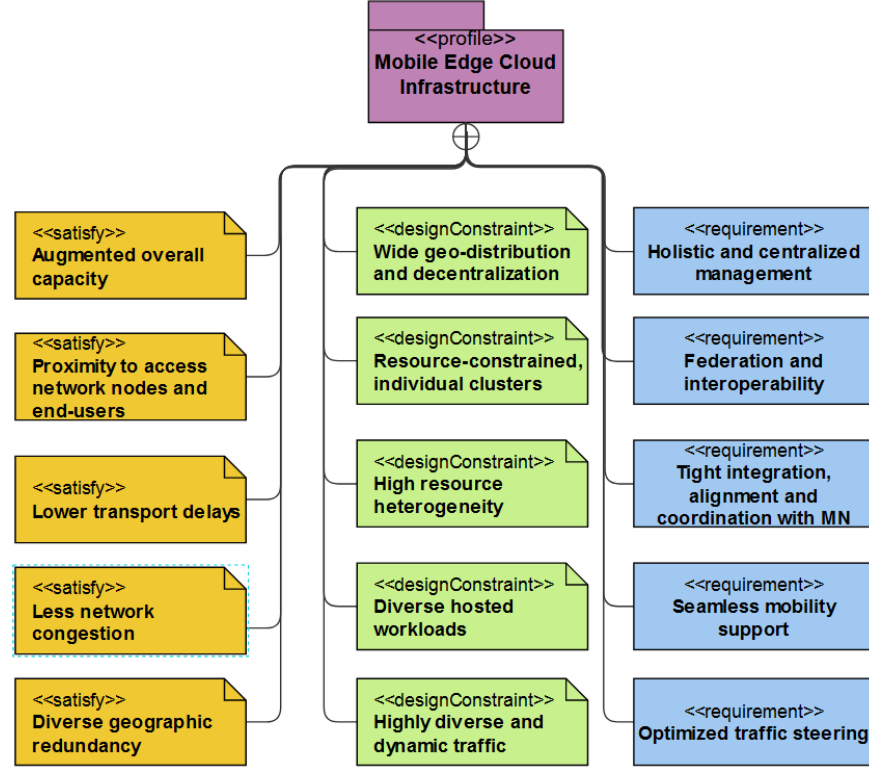


Figure 4: Evaluation Criteria: Characteristics and requirements of Mobile Edge Cloud infrastructure.

mechanisms, SM offers functionalities such as traffic performance monitoring, distributed request tracing, endpoint health checks and outlier detection (see Figure 5).

As condition/options of control, SM supports timeouts and failover/fallback options for circuit breaking. Traffic policies can be specified and enforced in a hop-by-hop basis, either by considering request origin information or destination information (see Figure 5). For origin-oriented conditions, Istio for example supports options such as origin service labels, origin namespace, session information and request parameters or headers. For destination-oriented conditions, Istio supports options such as destination service labels, destination namespace, request endpoint and request port.

#### 4.2 Design Drivers and Non-functional Characteristics

SM enables the reduction of duplicated code implementing common logic for traffic management based on the sidecar proxy pattern. The sidecar proxy pattern allows traffic management to be application agnostic, i.e., agnostic to the language in which the application is developed and, to some extent, the logic of the application itself. In contrast, a proxy-less approach for SM is proposed in Subhraveti et al. [2017]. For cloud-native applications built on the microservice-based architecture, a SM provides a way to compose a large number of discrete services into a centrally managed application. SM constitutes a highly manageable infrastructure layer enabling service-to-service communication, a relevant aspect of network softwarization.

One of the main design drivers of well-adopted SM implementations is related to DevOps practices. SM allows the decoupling of concerns between the application developer and the application operator. In this way, the configuration and maintenance of the service-to-service communication is not tied to the application code itself. Furthermore, SM supports more advanced functionality for Continuous Integration - Continuous Delivery/Deployment (CI/CD) such as A/B testing, blue/green and canary deployments.

In terms of traffic control and observability, SM supports flexible configurability thanks to the fine granularity levels that are supported in the definition of policy-based control conditions and actions. Policies are defined by following the so-called match-action abstraction, in which once a condition of control is met, a pre-defined action is matched and applied. More than just a cloud-native architectural pattern for traffic management, SM provides platform-level abstractions for easing and unifying traffic management of microservice-based applications.

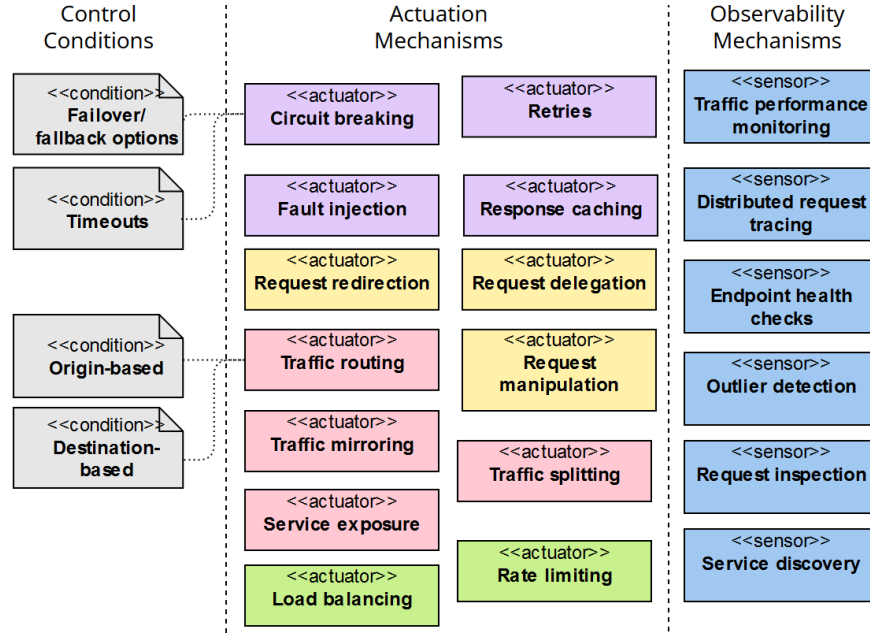


Figure 5: Functional characteristics of SoTA Service Mesh. Actuators in yellow and red are basic traffic control mechanisms, the ones in purple are related to reliability concerns, and the ones in green are related to bandwidth concerns.

Moving into more architectural matters, SM management, control and enforcement logics are decoupled into a Management Plane (MP), a Control Plane (CP) and a Data Plane (DP), respectively (see Figure 2). The MP, a.k.a. operational plane, is in charge of managing multiple SMs in a federated way. The MP can provide functionalities such as unified configuration and observability; discovery, interoperability and federation of multiple and heterogeneous SMs; and lifecycle management for SM and its applications, all under a single API.

The CP manages and configures the different components of the DP. It allows validation, ingestion, processing and distribution of configuration policies. It also monitors policy updates and propagates them to the DP components during runtime. The DP is usually composed by a set of lightweight proxies that are automatically injected and deployed along microservice instances, i.e., the sidecar proxy pattern. In more distributed deployment models, ingress, egress and/or east-west gateways are also part of the DP. DP components mediate communication on behalf of microservices and perform enforcement of traffic policies.

Actuation logic is generally applied in an on/off basis. Recent research proposals study softer/smoothier actuation mechanisms Sedghpour et al. [2021]. In terms of communication protocol support, SM implementations have limited support for Layer 4 / Layer 7 (L4/L7) communication protocols, i.e., L4 TCP, and L7 HTTP and gRPC.

As for network virtualization and abstraction matters, SM rely on the concept of network overlays such as the Container Network Interface (CNI). Furthermore, due to the sidecar proxy pattern, the traffic control and observability logic is run as a coupled process in user-space.

In terms of extensibility, Envoy-based DP components support WebAssembly (WASM) based extensions allowing flexible programmability of the proxy logic. On the other hand, many of the observability features of SM are based on external addons that can easily be inserted or removed.

In terms of deployment flexibility, Istio is one of the more mature implementations since it considers different dimensions related to various deployment models<sup>5</sup>: i. the cluster, ii. the network, iii. the CP, iv. the SM and v. the tenancy dimensions. Figure 6 summarizes SM's design drivers and non-functional characteristics.

<sup>5</sup><https://istio.io/latest/docs/ops/deployment/deployment-models/>, Istio: Deployment Models, accessed 2021-10-20



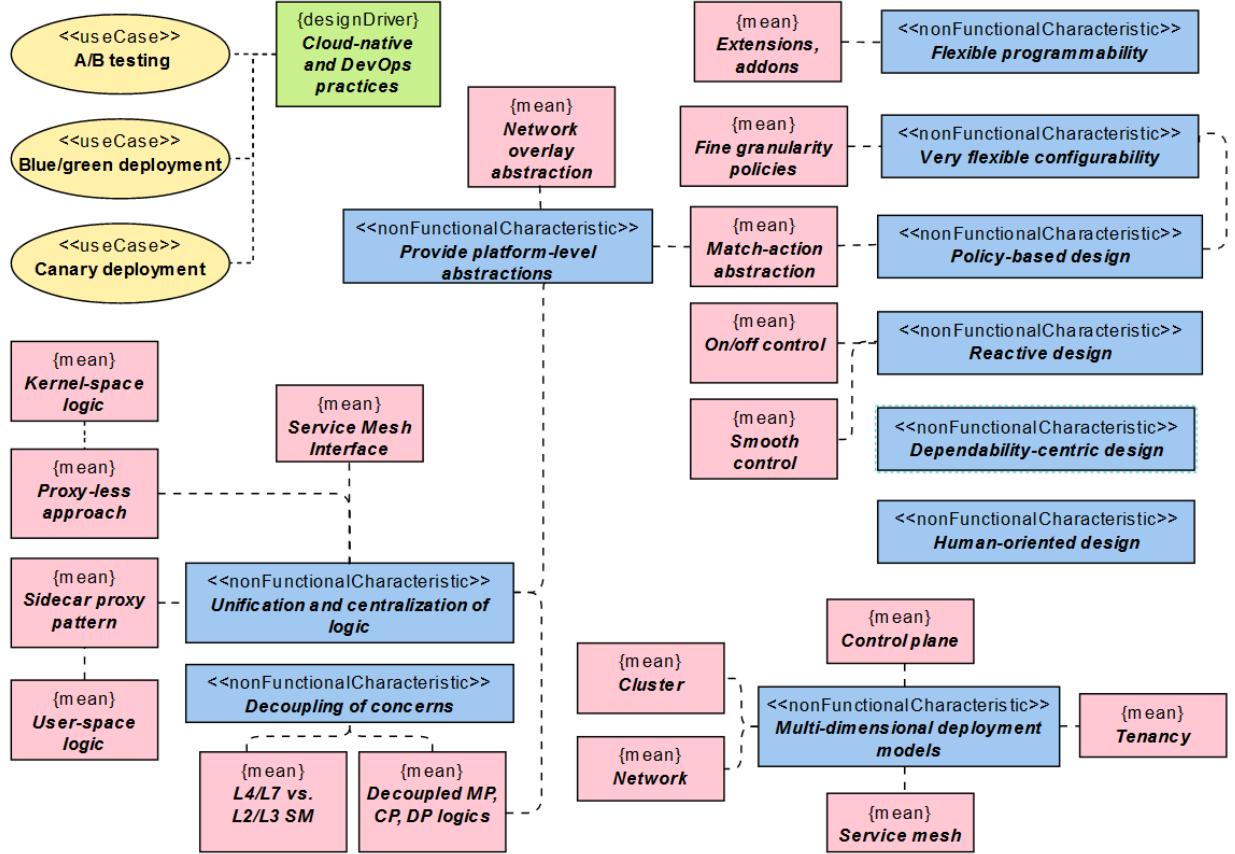


Figure 6: Design drivers and non-functional characteristics of SoTA Service Mesh

## 5 Qualitative Evaluation of Service Mesh-based Traffic Management

This section discusses the main results of the systematic and qualitative analysis we performed by comparing the evaluation criteria from MEC and its workloads presented in Section 3 vs. the characteristics and design drivers of SoTA SM presented in Section 4. This section also highlights limitations, tradeoffs and future research directions for the MEC-specialized SM we envision.

### 5.1 Efficient Service Mesh Architecture

Performance-demanding application workloads, in particular latency-sensitive applications, require the traffic management logic to be designed with minimum latency overhead. Similarly, resource-constrained individual edge clusters require minimum resource consumption. There are several aspects from the non-functional characteristics of SM that represent evident performance and resource costs.

**Service mesh interface** As we mentioned before, the SMI initiative aims at providing a minimum set of core features targeting today's SM use-cases in the context of generic IT application workloads running in more CC environments. In this sense, this initiative aims at providing a standard interface towards external cloud-native technologies, e.g., Kubernetes, to interact with different implementations of SM and to support federation across SM instances.

Nevertheless, current SM design lacks holistic or system-wide considerations related to the different domains and dimensions part of MEC; in particular, the design and standardization of a proper interface towards the MN to adequately implement functionalities such as end-to-end seamless mobility support or MN-aligned QoS assurance. To the best of our knowledge, we are the first ones to highlight this SM challenge.

**Communication patterns and protocols** SM is mainly oriented towards the fourth generation of application connectors Mendonça and Aderaldo [2021], in which application components mainly communicate via RESTful APIs.

The vast majority of generic IT application workloads today follow such communication pattern. However, this aspect represents a limitation in terms of supported communication patterns, which is also recognized by Kosińska, J. et al. Kosińska and Zieliński [2020]. Emerging application workloads may require more performant communication patterns such as the delayed approaches that work at the system call level.

SM supports connection-oriented and reliable protocols such as HTTP and TCP which are more appropriate for generic IT applications. This is due to the fact that traditional IT application workloads are mainly characterized by their reliability requirements. However, such protocols may not be adequate for latency-sensitive applications since they may suffer delay-causing issues such as Head-of-Line (HOL) blocking. Protocols with lower transport delays such as QUIC, UDP and RTP are preferred by latency-sensitive applications. Antichi, G., et al. also acknowledge this limitation Antichi and Rétvári [2020].

**Decoupled/split logic** There are intrinsic mechanisms part of the SM design intended to provide abstractions from the underlying system complexities. One of them is the decoupling of the traffic management into a MP, a CP and a DP, which is in part driven by the need of separation of concerns. Another related aspect is the fragmentation of SMs into L4/L7 class and L2/L3 class. On the other hand, SM relies on cloud-native network overlays. Such divide-and-conquer approaches have implications in terms of variable performance overheads that may not be appropriate for performance-demanding workloads; this issue is also identified by Antichi, G., et al. Antichi and Rétvári [2020].

**Sidecar proxy pattern** The sidecar proxy pattern is widely adopted by SM to provide unified control and observability logic with very fine granularity. However, this approach has been criticized due to the implicated latency overhead that is present especially for traffic exchange across co-located microservice instances. The sidecar proxy pattern also brings performance overheads due to context switching between user-space and kernel-space. Antichi, G., et al. also recognize this challenge Antichi and Rétvári [2020]. The added delays are exacerbated for longer and more complex microservice chains characteristic of MEC workloads.

The sidecar proxy pattern also incurs resource consumption overhead due to the fact that the traffic management logic is replicated at a per- microservice instance level. Such extra resource consumption may not be appropriate for edge clusters due to their resource constraints. As pointed out by Dobaj, J., et al. Dobaj et al. [2019], both latency and resource overheads represent inherent scalability concerns associated with the SM architecture.

Furthermore, the fact that proxies are injected per- microservice instance creates dependencies between the sidecar proxies and the application in terms of life cycle management, which may not be appropriate for highly dynamic management of traffic and SM infrastructure due to its limited flexibility.

On the other hand, since proxies implement both the actuation and instrumentation logic, this may represent a non-desired coupling between traffic control and observability, especially for automated traffic control, since such logics may have bi-directional performance implications.

The level of granularity required for traffic management of performance-demanding applications may not need to be the same than the one for generic IT applications. Especially, due to a tradeoff between the very fine granularity supported by SM for traffic control and observability vs. the incurred latency overhead of the sidecar proxies allowing such granularity levels.

In the case of Istio version 1.11.4, every couple of proxies adds about 2.65 ms to the 90th percentile latency. In the DP, an Envoy proxy uses 0.35 vCPU and 40 MB memory for 1000 req/s. In the CP, Istiod (Istio's daemon consolidating control plane functionality into a single binary) uses 1 vCPU and 1.5 GB of memory<sup>6</sup>.

Linkerd's own implementation of proxies, a Rust-based micro-proxy, is designed by considering performance requirements, thus service-to-service communication takes less than 1/5th the maximum extra latency taken by Istio while consuming 1/9th the memory and 1/8th the CPU when compared to Istio's DP<sup>7</sup>.

AppSwitch has been used as an Istio plugin to test performance enhancements. Results indicate an enhancement of over 18 times in the 50th percentile latency compared to vanilla Istio<sup>8</sup>. A similar proxy-less approach is recently starting to get adopted by Cilium<sup>9</sup> based on eBPF, a kernel technology supporting custom programs to be run in kernel-space. However, the aforementioned performance metrics for Istio, Linkerd and AppSwitch are not comparable since they

<sup>6</sup><https://istio.io/latest/docs/ops/deployment/performance-and-scalability/>, Istio Performance and Scalability, accessed 2021-11-28

<sup>7</sup><https://linkerd.io/2021/05/27/linkerd-vs-istio-benchmarks/>, Benchmarking Linkerd and Istio, accessed 2021-11-04

<sup>8</sup><https://istio.io/latest/blog/2018/delaying-istio/>, Delaying Istio with AppSwitch: Automatic application onboarding and latency optimizations using AppSwitch, accessed 2021-11-01

<sup>9</sup><https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh>, How eBPF will solve Service Mesh - Goodbye Sidecars, accessed 2021-12-10

are not performed under the same conditions nor consider conditions characteristic of MEC setups. Nonetheless, performance enhancements from the kernel-space approaches need to be carefully evaluated in large-scale and highly distributed environments.

Table 1 summarizes all of these challenges. To the best of our knowledge, we are the first ones to highlight the challenges in the table that do not have associated any reference.

Table 1: Limitations associated with efficient Service Mesh architecture

Functional/ Non-functional Characteristic	Limitations
Service mesh interface	Lack of holistic interface design for end-to-end integration between the MN and the EC domains
Communication patterns and protocols	Limited communication pattern support; mainly for the 4 <sup>th</sup> generation of application connectors Kosińska and Zieliński [2020] Lack of support for faster/telco-grade communication protocols and standards Antichi and Rétvári [2020]
Decoupled/split logic	Decoupled MP, CP and DP logic may introduce performance overheads due to the need of network-based communication across planes L2/L3 SM and L4/L7 SM represent a split approach of the communication stack which may introduce performance overheads Antichi and Rétvári [2020] Network overlay may introduce performance issues with variable delay overheads
Sidecar proxy pattern	Limited scalability and resource overhead due to replicated proxy logic per- microservice instance Dobaj et al. [2019] Per-hop latency overhead even for communication across co-located microservice instances Antichi and Rétvári [2020] Context-switching overhead since sidecar proxy logic is run in user-space Antichi and Rétvári [2020] Latency and resource overhead due to inadequate granularity level in control and observability logics Dependent and inflexible life-cycle management of proxies and microservice instances Performance coupling between observability and actuation logics since both are implemented in a common sidecar proxy Lack of quantification of performance costs associated with sidecar vs. sidecar-less approaches on large-scale applications hosted on edge cloud

## 5.2 Differentiated QoS Assurance Support in Service Mesh

Performance demands of emerging applications require more efficient traffic steering mechanisms with stronger QoS assurance support. The fact that SM is not designed for deep EC is reflected in the lack of awareness, alignment, coordination and integration with the QoS assurance mechanisms proposed for the MN domain. Antichi, G., et al. also identify a general lack of support for QoS assurance Antichi and Rétvári [2020].

**Dependability-centric traffic control** In terms of traffic control, most of the condition-based traffic actuation mechanisms supported by SM (see Figure 5) are intended to address reliability requirements of generic IT applications. This represents a good initial set of features for applications with strong dependability needs such as mission-critical applications. However, the traffic management needs from bandwidth-demanding applications and, most of all, the ones from latency-sensitive applications are not considered by such traffic control mechanisms. To the best of our knowledge, we are the first ones to highlight this SM challenge.

There is a lack of holistic and differentiated QoS assurance mechanisms for the different types of application requirements of MEC workloads in alignment with the QoS assurance mechanisms of the MN and with awareness of the dynamic traffic performance through the MN path. Ways to address such limitations are end-to-end SFC for MEC application workloads similar to the ones proposed in Dab et al. [2020a]Dab et al. [2020b], but rather based on L4/L7 SM. Other more appropriate actuation mechanisms may include prioritization of traffic and support for dynamic traffic shaping.

**General-purpose multi-tenant isolation** SM considers both hard- and soft- multi-tenant isolation requirements of generic IT workloads in its deployment models and in its condition-based traffic control logic. However, there is no thorough consideration regarding isolation of the delivered performance to avoid performance implications across workloads with diverse characteristics and requirements. We consider that support for performance-oriented multi-tenant isolation similar to the concepts of network slicing would be needed in the MEC context. To the best of our knowledge, we are the first ones to highlight this SM challenge.

**Zero-downtime rolling updates** Today's SM provides a set of actuation mechanisms such as traffic mirroring, traffic splitting and traffic redirection (see Figure 5) intended to be used for zero-downtime rolling updates across application versions<sup>10</sup>. We envision that such actuation mechanisms can be exploited with a different purpose in MEC workloads. As mentioned before, one of the main challenges of MEC is the highly dynamic traffic conditions such as the ones generated upon mobility events. Actuators originally designed for providing zero-downtime application rollouts can be leveraged to provide seamless mobility support with minimum QoE degradation. To the best of our knowledge, we are the first ones to highlight this SM opportunity.

**Human-oriented traffic observability** In terms of traffic observability, the instrumentation and data collection mechanisms implemented in the SM are mainly intended to be used by human system operators. Nevertheless, such proposed mechanisms may not necessary be appropriate for automated traffic management. To the best of our knowledge, we are the first ones to highlight the following SM challenges.

The per-hop, per-endpoint granularity associated with traffic performance metrics may not be appropriate for differentiated QoS assurance since it does not consider aggregated performance estimations at the level of end-to-end service chains.

Furthermore, the frequency of performance metric collection may not be fast enough to appropriately detect and react upon fast changing traffic load conditions. In the case of Prometheus, for example, it is recommended not to have a scraping interval lower than 15s due to Kubelet's resource usage metric resolution. Envoy's traffic performance metrics can be flushed into stats sinks with a minimum interval of 1ms; however, it is unknown if Prometheus would scale well upon high traffic load and large-scale deployments.

To perform automated traffic control, other system metrics need to be considered from different parts of the application and the underlying execution environment Kosińska and Zieliński [2020]. On the other hand, the way in which traffic observability is provided via external addons, as in Istio, offers poor configurability of the observability logic.

Table 2 summarizes all of these challenges. To the best of our knowledge, we are the first ones to highlight the challenges in the table that do not have associated any reference.

### 5.3 Autonomous Service Mesh

Intelligence, automation and autonomy are desired characteristics of cloud-native traffic management approaches to cope with the strong traffic performance guarantees MEC workloads require. There is a need for having more abstracted, automated and autonomous mechanisms to efficiently adapt not only the ongoing traffic but also the SM architecture itself to the highly dynamic conditions characteristic of emerging applications and their underlying execution environment.

**Flexible configurability** Today's SM offers very fine granularity in terms of traffic control policies and it also supports many dimensions of configurability in their deployment models to provide the very flexible configurability required by generic IT use-cases. Nevertheless, SM frameworks are still considered complex to operate; such levels of flexibility hamper their adoption due to the inherent degrees of complexity, as mentioned by Dobaj, J., et al. Dobaj et al. [2019].

<sup>10</sup><https://blog.sebastian-daschner.com/entries/zero-downtime-updates-istio>, Zero-Downtime Rolling Updates With Istio, accessed 2021-09-01

Table 2: Limitations associated with QoS assurance support in Service Mesh

Functional/ Non-functional Characteristic	Limitations
Dependability-centric traffic control	Limited dependability-centric design with focus on reliability requirements
	Unawareness of diverse application performance requirements
	Lack of differentiated traffic routing logic based on application requirements
	Lack of actuation, conditions of control and observability mechanisms for other purposes than merely reliability support
	Lack of traffic shaping mechanisms for bandwidth-demanding applications
	Lack of traffic prioritization mechanisms for latency-sensitive applications
General-purpose multi-tenant isolation	No support for performance-oriented, multi-tenant isolation
Zero-downtime rolling updates	Lack of seamless mobility support with minimum QoE degradation
Human-oriented traffic observability	Intended to be for human operator's visualization
	Inappropriate level of granularity in performance metrics, not at the service chain level required for QoS assurance
	Lack of end-to-end performance estimation
	Limited frequency of metric collection, not appropriate for very dynamic changes in metrics values
	Inflexible and cumbersome configurability of metric collection due to poor integration with external addons
	Performance metrics are limited to traffic monitoring and tracing; they need to be complemented with system performance metrics Kosińska and Zieliński [2020]

Despite all efforts to make traffic management easier, we consider that current levels of abstractions are just the first step towards higher levels of autonomy in SM. Automation and autonomy could offer higher levels of abstraction with more human-intuitive mechanisms than the ones provided by policy-based traffic management, for example.

**Complexity abstractions** The match-action abstraction has historically been used by human network operators. Such traffic control mechanism is more suitable for rule-based systems dealing with relatively less dynamic and simpler traffic conditions. However, this mechanism may not be appropriate to build more advanced, automated and autonomous traffic management approaches which need to address the strong performance demands of future workloads under highly dynamic conditions.

Furthermore, different approaches used for complexity abstractions such as network overlays and the sidecar proxy pattern imply certain performance costs. There is a need to quantify the associated performance cost vs. performance gains of automated/autonomous traffic management mechanisms. To the best of our knowledge, we are the first ones to highlight the mentioned SM challenges.

**Decoupled policy propagation** The SM CP takes care of propagating policy updates to the different components of the DP to deal with changes in traffic control stipulated by human operators. However, such decoupled mechanism may not be able to support frequent policy reconfigurations associated with automated/autonomous traffic management in scalable and performant ways. Highly dynamic reconfigurations may lead to network bottlenecks when the CP propagates frequent updates across DP components. To the best of our knowledge, we are the first ones to highlight this SM challenge.

**Human-oriented management plane** Today's SM MP only takes care of SM operational matters with very poor levels of automation and autonomy. As recognized by Antichi, G., et al. Antichi and Rétvári [2020], we also envision

Table 3: Limitations associated with autonomous Service Mesh

Functional/ Non-functional Characteristic	Limitations
Flexible configurability	Complex to operate which tamper their adoption Dobaj et al. [2019]
	Fine granularity level in configuration imply performance overheads
	Lack of higher levels of abstraction e.g., intent-driven configuration
Complexity abstractions	Match-action abstraction limited to rule-based systems and not appropriate for autonomous management
	Complexity abstractions imply performance costs; there is a need to evaluate performance cost vs. performance gain of higher abstraction levels
Decoupled policy propagation	Not designed to support frequent policy reconfigurations in scalable and performant ways
Human-oriented management plane	SM management plane limited to human-oriented operational matters
	Lack of self-adaptable functionality and architecture Mendonça and Aderaldo [2021] Antichi and Rétvári [2020]
Reactive traffic control	Actuation and condition mechanism limited to on/off, lack of softer/smooth control mechanisms Mendonça and Aderaldo [2021] Sedghpour et al. [2021]
	Lack of analytics and inference layers with more predictive and proactive logics Li et al. [2019]

a SM capable of supporting architectural adaptability and on-the-fly configuration of communication protocols and patterns in response to highly dynamic and diverse requirements of MEC workloads.

**Reactive traffic control** Traffic control mechanisms supported by today’s SM implementations are reactive; the actuation logic is triggered upon the detection of a set of conditions in an on/off fashion. There is a lack of proactive and smooth actuation mechanisms which may be enabled by the introduction of, for example, analytics and inference layers Li et al. [2019]. Mendonça, N., et al. also state that none of the current SM solutions support general-purpose self-adaptation capabilities Mendonça and Aderaldo [2021]. Saleh Sedghpour, M. R., et al. also identify the lack of adaptable traffic control mechanisms such as smooth circuit breaking actuation logic Sedghpour et al. [2021].

Table 3 summarizes all of these challenges. To the best of our knowledge, we are the first ones to highlight the challenges in the table that do not have associated any reference.

## References

- Ali J. Fahs and Guillaume Pierre. Proximity-aware traffic routing in distributed fog computing platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 478–487, 2019. doi:10.1109/CCGRID.2019.00062.
- 3GPP. Study on enhanced application architecture for enabling edge applications. Technical Report (TR) 23.700-98, 3rd Generation Partnership Project (3GPP), 10 2021. URL <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3920>. Version 0.3.0.
- D. Subhraveti, S. Goli, S. Hallyn, R. Chamarthy, and C. Kozyrakis. Appswitch: Resolving the application identity crisis. In *arXiv*, November 2017.
- Eyal Cidon, Sean Choi, Sachin Katti, and Nick McKeown. Appswitch: Application-layer load balancing within a software switch. In *Proceedings of the First Asia-Pacific Workshop on Networking, APNet’17*, page 64–70, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352444. doi:10.1145/3106989.3106998. URL <https://doi.org/10.1145/3106989.3106998>.
- Jürgen Dobaj, Markus Schuss, Michael Krisper, Carlo Alberto Boano, and Georg Macher. Dependable mesh networking patterns. In *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLop ’19*, New

- York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362061. doi:10.1145/3361149.3361174. URL <https://doi.org/10.1145/3361149.3361174>.
- Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. Service mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225, April 2019. doi:10.1109/SOSE.2019.00026.
- Gianni Antichi and Gábor Rétvári. Full-stack sdn: The next big challenge? In *Proceedings of the Symposium on SDN Research, SOSR '20*, page 48–54, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371018. doi:10.1145/3373360.3380834. URL <https://doi.org/10.1145/3373360.3380834>.
- Nabor C. Mendonça, David Garlan, Bradley Schmerl, and Javier Cámara. Generality vs. reusability in architecture-based self-adaptation: The case for self-adaptive microservices. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18*, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450364836. doi:10.1145/3241403.3241423. URL <https://doi.org/10.1145/3241403.3241423>.
- Mohammad Reza Saleh Sedghpour, Cristian Klein, and Johan Tordsson. 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 '21*, page 4–10, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383363. doi:10.1145/3447851.3458740. URL <https://doi.org/10.1145/3447851.3458740>.
- J. Kosińska and K. Zieliński. Autonomic management framework for cloud-native applications. *J Grid Computing*, 18: 779–796, 2020. URL <https://doi.org/10.1007/s10723-020-09532-0>.
- Lars Larsson, William Tärneberg, Cristian Klein, Maria Kihl, and Erik Elmroth. Adaptive and application-agnostic caching in service meshes for resilient cloud applications. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 176–180, June 2021. doi:10.1109/NetSoft51509.2021.9492576.
- Nabor Mendonça and Carlos Aderaldo. Towards first-class architectural connectors: The case for self-adaptive service meshes. In *Brazilian Symposium on Software Engineering, SBES '21*, page 404–409, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390613. doi:10.1145/3474624.3477072. URL <https://doi.org/10.1145/3474624.3477072>.
- Boutheina Dab, Ilhem Fajjari, Mathieu Rohon, Cyril Auboin, and Arnaud Diquélou. Cloud-native service function chaining for 5g based on network service mesh. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, June 2020a. doi:10.1109/ICC40277.2020.9149045.
- Boutheina Dab, Ilhem Fajjari, Mathieu Rohon, Cyril Auboin, and Arnaud Diquélou. An efficient traffic steering for cloud-native service function chaining. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 71–78, Feb 2020b. doi:10.1109/ICIN48450.2020.9059340.