Cluster Computing (2024) 27:8965–8982
https://doi.org/10.1007/s10586-024-04344-3

ServiceNet: resource-efficient architecture for topology discovery in large-scale multi-tenant clouds

Angel Gama Garcia[1] · Jose M. Alcaraz Calero[1] · Higinio Mora Mora[2] · Qi Wang[1]

Received: 30 November 2023 / Revised: 18 January 2024 / Accepted: 5 February 2024 / Published online: 13 April 2024

Abstract
Modern computing infrastructures are evolving due to virtualisation, especially with the advent of 5G and future technologies. While this transition offers numerous benefits, it also presents challenges. Consequently, understanding these complex systems, including networks, services, and their interconnections, is crucial. This paper introduces ServiceNet, a groundbreaking architecture that accurately performs the important task of providing understanding of a multi-tenant architecture by discovering the complete topology, crucial in the realm of high-performance distributed computing. Experimental results have been carried out in different scenarios in order to validate our approach, demonstrating the effectiveness of our approach in comprehensive multi-tenant topology discovery. The experiments, involving up to forty tenant, highlight the adaptability of ServiceNet as a valuable tool for real-time monitoring in topology discovery purposes, even in challenging scenarios.

Keywords Topology discovery · Distributed computing · Resource management · Multi-tenancy · 5G networks

1 Introduction

In recent years, modern computing infrastructures have gone through a significant transformation encompassing both hardware and software. This transformation has been driven by a pursuit of increasing performance and efficiency. The emergence of 5G and beyond such as 5G Advanced networking towards 6G marks the beginning of a new era, promising remarkable capabilities such as ultra-high-speed connectivity, extremely low latency, softwarisation and virtualisation of infrastructures and services, and the seamless integration of the digital and physical worlds. However, these promises bring along substantial challenges such as an improved system comprehension, communication or monitoring [1].

The backbone of our digital age now relies on large-scale multi-tenant infrastructures. These infrastructures host a wide range of services and applications that play a crucial role in our daily lives. They encompass everything from cloud-based business applications to edge computing in different infrastructures. The size and complexity of these infrastructures make conventional management methods less and less adequate over time [2]. As these multi-tenant infrastructures continue to grow and evolve, the demand for solutions for efficient and scalable service discovery is vital to facilitate service management in such complex networking and computing environments.

While traditional infrastructure management often involves static configurations and manual intervention, this approach is increasingly being challenged. However, it is important to note that, even though there have been significant advancements in automation technologies and their adoption, the transition towards automation in this field is not yet complete. This can be due to a variety of factors, including the complexity of existing systems, the need for significant investment in new technologies, and the challenges associated with changing established operational procedures and workforce skills. Today's applications and services are dynamic in nature, and resources need to scale rapidly to meet fluctuating demands. This makes manual intervention impractical and prone to errors. Infrastructure

✉ Jose M. Alcaraz Calero
jose.alcaraz-calero@uws.ac.uk

[1] University of the West of Scotland, High Street, Paisley, Renfrewshire PA1 2BE, Scotland

[2] Universidad de Alicante, Carr. de San Vicente del Raspeig, 03690 Alicante, Spain

operators face a daunting challenge: they must ensure the smooth operation of numerous services in a vast multi-tenant environment while adapting to real-time changes.

Moreover, service disruptions in these infrastructures can have far-reaching consequences. They affect not only the businesses and organisations that depend on them, but also the end-users who rely on uninterrupted access to digital services. Consider, for instance, a scenario where a cloud service outage occurs, leading to the temporary suspension of critical business applications. Initially affecting company revenues and operations, these disruptions can subsequently reach end-users, disrupting their daily activities. Therefore, the need to develop advanced tools and methodologies for service discovery is driven not only by technological progress but also by the social and economic impact of infrastructure failures. This implies that aside from the technical disruptions caused by service outages, there are profound consequences on economic stability, market dynamics, and various sectors.

Traditionally, monitoring tools have focused on host-level services. Nevertheless, with the new digitalisation era, rises a new challenge that lies in understanding the behaviour and performance of resources at deeper and wider levels [3]. The fundamental issue is clear: ensuring service reliability, performance and security in softwarised and virtualised environments requires new approaches that go beyond the traditional techniques. This shift has given rise to new terms like 'slicing,' applied to both network and computer services [4, 5]. As these environments continue evolving, the need for solutions increases and new challenges are emerging. These challenges can be categorised into two main areas: Network Topologies, and Operation and Performance. The first category, Network Topologies, addresses issues such as the integration of both physical and logical network topologies, which is crucial for creating a comprehensive view of service landscapes. The orchestration of overlay topologies is another key challenge, as it entails coordinating the discovery of various overlay network structures to allow a holistic understanding of service deployments. The need of discovery improvement in the area of multi-cloud environments should be emphasised [6]. The second category, Operation and Performance, deals with challenges related to data high performance availability, its visualisation and cross-domain interoperability. Data visualisation is essential for effectively presenting the discovered service topologies, while ensuring seamless interoperability across different domains. These challenges collectively form the foundation for addressing the complexities of service topology discovery in cutting-edge network environments.

In response to these challenges, this work delves deep into the topology discovery in terms of a multi-tenant system in 5G and beyond networks, driven by the accelerating trend of virtualisation and the growing complexity of service management requirements in this next-generation network paradigm. Accordingly, the paper presents the following technological contributions (Table 1).

- Creation of an innovative resource-efficient architecture called ServiceNet to achieve a real-time multi-tenant topology registry and further visualisation. It offers a robust tool for managing and visualising topology information across the large-scale system.
- New capabilities to allow gathering both network and service topological information in complicated 5G and beyond networking environments. Achieving a fully-connected information space in order to visualise not only the services but also their socket connection and where they belong in the host.
- Experiments with different multi-tenant configurations and empirical results on the testing and validation of the functionality of the proposed architecture.

The remaining of this paper is organised as follows. Section 2 reviews the current state of the art in the field of service discovery. Section 3 describes the proposed architecture developed to achieve the services discovery in the field of multi-tenant architectures. Section 4 provides details on the design and implementation of the proposed architecture. Section 5 presents the validation and results of the proposed approach. Different experiments are carried out to show how the architecture performs in different scenarios. Finally, Sect. 6 concludes the paper with a brief summary of the obtained results and their implications.

2 Related work

Topology discovery within complex IT infrastructures is a pivotal undertaking, particularly in the domain of large-scale multi-tenant environments. The ability to automatically identify the services, locate them and finally find the communication between them is essential for a better understanding of the infrastructure as well as a real-time responsiveness. In this section is carried out a study of the related work in this field, with the aim to understand both the strengths and weaknesses within the existing research.

Topology Discovery is a relative new area. In response to the challenge on obtaining an overall picture of the whole network and service topology, we conducted literature searches with different configurations: Search 1: ("All Metadata":"Topology discovery") AND ("All Metadata":multi-tenant), Search 2 ("All Metadata":"Topology discovery") AND ("All Metadata":tenant), and Search 3 ("All Metadata":"Service topology") AND ("All Metadata":discovery). It is noted that results are filtered between 2005 and 2023. In Table 2, we present the results searching

Table 1 Comparison of this paper (ServiceNet) with related work

Criteria	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	ServiceNet
Discover Network Topology	✔	✔	✔	✔	✘	✔	✔	✔	✔
Discover 5G Topology	✔	✘	✘	✘	✘	✘	✔	✘	✔
Discover sockets	✘	✘	✘	✘	✘	✘	✘	✔	✔
Discover service	✘	✔	✔	✔	✘	✔	✔	✔	✔
Connection socket-service	✘	✘	✘	✘	✘	✘	✘	✔	✔
Distributed discovery	✔	✔	✔	✘	✔	✔	✔	✔	✔
Container network topology	✔	✘	✘	✘	✘	✘	✘	✘	✔
Container socket topology	✘	✘	✘	✘	✘	✘	✘	✘	✔
Container service topology	✘	✘	✘	✔	✘	✘	✘	✘	✔
Multi-tenant topology	✘	✘	✘	✘	✘	✘	✘	✘	✔
Fault tolerance	✘	–	✘	✘	✔	✔	✔	✔	✔
Automatic service registration	✘	✘	✔	✘	✘	✔	✔	✘	✔
Real-time gathering	✔	–	✔	–	✔	✔	✔	✔	✔
Real-time topology visualisation	✘	✘	✘	✘	✘	✘	✘	–	✔

articles and conferences from three different online databases, including Taylor & Francis, IEEE Xplore, ScienceDirect and Springer. The column "Data" introduces the number of contributions in the search, and "Cited" the papers that have been found related to our work and consequently commented. The table indicates the level of novelty of the proposed work in the field of topology discovery in multi-tenant infrastructure, highlighting the overall enlarged capabilities of the proposed and validated work in this paper, beyond the state of the art.

In the realm of service management, we can divide the service registry methods in four, each with its own set of merits and drawbacks. First method involves services being registered directly by the software that manage and deploys them, can be found in tools such as Juju [16], Capistrano [17], Scalr [18], and Puppet [19]. While offering reliability, this approach often necessitates a substantial automation system, making it less practical for organisations lacking advanced automation capabilities. Method two hinges on API-based self-registration, where services are autonomously registered by themselves, includes tools like RMI registry [20], and CORBA [21]. However, implementing self-registration may demand software modifications, posing challenges in terms of compatibility. In Method number three, administrators manually register services, exemplified by systems like OpenStack [22]. This method is knowledge-intensive as well as time-consuming, relying heavily the expertise of the administrator. Notably, in our work, we introduce the method number 4, a novel approach for automatic software-based registration, a method distinct from existing literature. In this approach, services are registered automatically, mitigating the need for manual intervention and offering enhanced efficiency and reliability in service management.

In the landscape of service discovery protocols and methodologies, there are common service discovery

Table 2 Literature review search

	Search 1		Search 2		Search 3	
	Data	Cited	Data	Cited	Data	Cited
Taylor &Francis	1	0	1	0	0	0
IEEE Xplore	0	0	3	[6]	4	[8]
ScienceDirect	21	[15]	40	[15]	31	–
Springer	11	[7, 14]	19	[7, 14]	38	[14]

protocols worth commenting, as seen in [23]. They are basically used to enable applications and microservices to locate different components on a network. Some examples include Bluetooth Service Discovery Protocol (SDP) [24] used for discovering Bluetooth devices and services and DNS Service Discovery (DNS-SD) [25], which enables automatic discovery of computers, devices, and services on an IP network. These protocols play a significant role in automating the discovery of services and reducing manual configuration tasks, which is crucial for efficient network management.

mDNS zeroconf (zero-configuration) protocols allow automatic device discovery and connections within a network, removing the need for manual configuration. Avahi [26] is an open-source implementation of zeroconf technologies for Unix-like systems. It works as an mDNS/DNS-SD responder. It enables automatic service discovery simplifying network setups by just plugging the computer into a network and instantly viewing other services such as printers, files being shared, media, and so on. Compatible technology is found in Apple, branded Bonjour [27]. It integrates mDNS and DNS-SD protocols into Apple ecosystem and other platforms. It simplifies device and service discovery on local networks, fostering easy connectivity between Apple devices.

Now, let's delve into several papers that address various aspects of topology discovery. Each of them is compared in Table 1 having into account criteria regarding our cornerstones. While each one approaches a different goal, they share a common objective of improving network comprehension. In [28], Duan and Lu presents an architecture for discovery on service-oriented networks, it highlights the importance of virtualisation within networking and its service-oriented nature. Sanchez et al set a good starting point in network topology discovery in [7], showing empirical validation experiments where they get the topology and monitor a 5G multi-tenant network. In [8], is introduced a novel approach to service composition in dynamic ad hoc environments. It addresses the dynamic nature of service discovery, emphasising the need for flexibility, reduced communication overhead, and efficient resource utilisation in such environments. The publication [9] proposes a scalable and self-configuring peer-to-peer (P2P)-based architecture for large-scale IoT networks, focusing on automated service and resource discovery mechanisms in IoT networks. It shows a 2D grid with the smart objects deployments. Mathews et al present an interesting discussion of fault management with cross-layer service topology in [10]. It uses Gephi [29] to visualise the topology discovery and identifies the need to discover the cross-layer cloud service topology to deal with service disruptions. This work presents valuable insights and methods in the field of service topology visualisation, but it lacks of more general network comprehension.

Closed-source tools related to our work can be found from various sources [11–13] and are subsequently introduced. An interesting product auto-defined as Network Topology Mapper (NTM) [11] created by SolarWinds, carries out an automated device discovery and mapping using ICMP, SNMP, WMI, CDP, VMware, Microsoft Hyper-V, and more. It limits its topology to just networking information. There are more tools that delve deeper in the topology discovery, [12] is a software that enables network discovery, with a primary focus on network configuration management. It uses network protocols such as SNMP, WMI, and REST APIs to gather information. Another software solution is [13], used to monitor industrial IT and IoT infrastructures integrating technologies such as Ping, SNMP, WMI, SSH, HTTP requests, and different flow protocols (IPFIX, jFlow, sFlow, NetFlow). Although it comments Virtual Machine (VM) monitoring, it does not go deep in the topology or services being run. It is important to note that due to their closed-source nature, there is limited publicly available information and testing on the three commented software [11–13].

Wei et al. in [14], achieve a very complete Topology Discovery in the field of multi-cloud environments. In their work it is proposed a framework that gathers topology information by a "topology engine" and then it is shown by a visualiser, being able to draw an overall application distribution graph. As can be seen in the table, the paper is highly aligned with our work, with some caveats, as it is the only one accomplishing the 'Discover Sockets' and 'Connection Socket-Service' criteria, making it relevant for our focus on multi-tenant environments."

Cummins [30] introduces the definition of modern service as *all the activities that an organisation does to plan, design, deliver, operate, and control the applications in an enterprise. It includes the people who do the work, processes that define what work is needed and how it is done, and tools to enable and support these activities. Applications are monitored to ensure availability and performance according to service level agreements (SLAs) or service level objectives (SLOs)*. Benefits of service management are also commented. Along them are found an improved effectiveness, increased operational reliability and agility, operational efficiency by using real-time analytics, application's performance and higher management of risks and threats.

A similar work can be found also from IBM where Averdunk investigates in the principles of cloud service management and operations [31]. It is underscored the importance of several key principles in a production environment that will help to adopt microservices-based applications. It is commented the importance of service discovery, workload based on resource requirements, self-healing, and even the need of real time monitoring metrics beyond CPU, memory and disk space. Our research extends the principles commented by Cummins to industrial environments. We aim to improve the digitalised service management practice by real-time monitoring the topology of the services. Thereby enhancing network understanding and reliability in these critical contexts as important keys mentioned in [31]. The lack of available tools in the literature to perform both service and network topology discovery in multi-tenant environment has been the main motivation of this research work.

Moreover, in current software there is a trend in multi-domain, specially in 5G networks. The majority of them operate within multi-instance frameworks, often executing across tenats and multi-cloud providers. Our approach stands apart by operating from an external point consuming resources in a singular instance rather than a multi-instance, because it is only executed in the host. This allows for resource optimisation. Being also useful for large scale deployments together with our distributed architecture.

2.1 Findings

This section intends to clarify the key aspects of our approach that contribute significantly with relation to the current state of the art. Table 1 delineates the different key points achieved in comparison to work found in the topology discovery field. The network topology is the feature that we have found more times [7–10, 12–14], as well as the distributed configuration and Real-Time gathering [7–9, 11–14]. In contrast, it has been difficult to see discovery of 5G topology networks, this is a key aspect of our approach achieved thanks to the application of RIA, an agent found in the work done by Sanchez et al in [15]. Leveraging this, we can discover 5G and beyond network topologies alongside their interconnected services and sockets within virtualised components.

The literature review has encountered challenges in discovering socket, only done by [14]. Notably, no work on container socket topology has been found, being a key point in our tenant discovery as we treat tenants as containers inside the multi-tenant. While discovery of services is seen in various contributions [8–10, 12–14], there is a clear lack when talking about tenant or containers. Only one performs a topology discovery in this context, using it to discuss further implementation of accurate fault management techniques [10].

Some of these also implement fault tolerance techniques ensuring secure discovery [11–14]. The registration methods is previously commented in this section, we follow an automatic registration without human intervention or client/server acceptance, something seen in [9, 12, 13]. Also, regarding visualisation it is true that some tools allow us to visualise the topology but they are limited to rather network or services, without going in deep as our approach does with network, services, sockets and containers. Overall, we can confirm that we have found different contributions that are aligned to our approach, but is has been identified a clear lack in the 5G and container discovery, and so in its further visualisation.

3 Proposed architecture for service topology discovery

In the context of this work, we define the term “Service” as a software component within the computer system that performs a defined task. Often defined as “micro-service” [14]. Services may include a wide range of elements such as processes, which can encompass threads and daemons. These components are the responsible for carrying out specific operations using and managing computer resources as well as providing communication capabilities. The interconnectivity and communication capabilities are exemplified in our service topology discovery. Furthermore, ‘Socket,’ is defined as the communication endpoints for services running on a device. Each socket is uniquely identified by a combination of the device IP address and a specific port number. This allows for services to transmit and receive information using different protocols such as TCP or UDP (Figs. 1, 2).

Topology Discovery Agent (TDA) is the core component of the architecture. It is responsible for discovering both network and services on the host and the containers. It leverages the use of the component called Resource Inventory Agent (RIA) presented in [15] by Sanchez et al. which is responsible for for discovering the following information: (i) Physical Machine and its logical and virtual network interfaces, (ii) VMs and their virtual network interfaces, (iii) Containers and their network interfaces, (iv) Software switches, (v) Specialised Physical Devices such as Software-Defined Radio (SDR) and their network interfaces, (vi) Interconnection between network interfaces, and (vii) Multi-tenant information of VMs. It is important to provide information of where are the services located and its relationship within the infrastructure. Due to this, TDA goes further and gathers services information: (i) Host Services, (ii) Containers Services, (iii) Host Network Sockets, (iv) Container Network Sockets, (v) Host Service to Host Network Socket connection, (vi) Container Service to Container Network Socket connection.

Tenant host is formed by the physical computer or Virtual Machine, depending on the scenario, which is shared among multiple users, client organisations, or any kind of tenant. In the context of 5G/6G virtualisation is one of the main keys, which leads us to run multiple

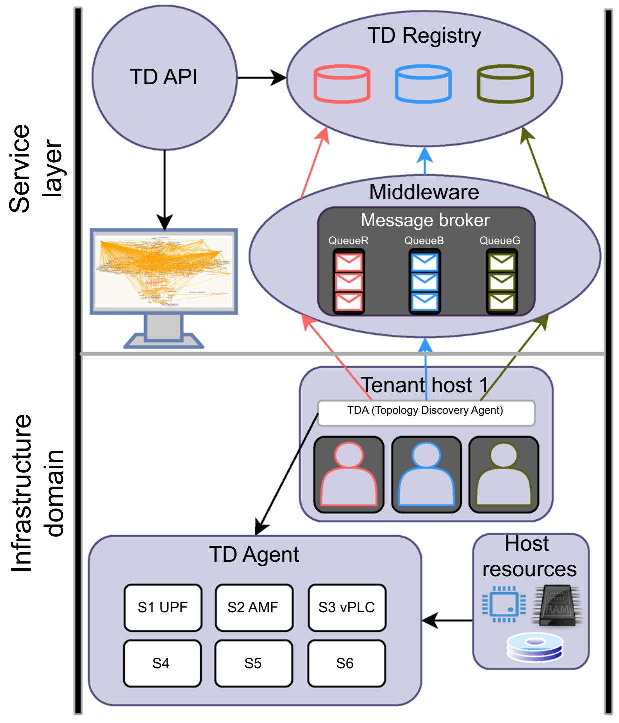


Fig. 1 ServiceNet architecture design for service topology discovery

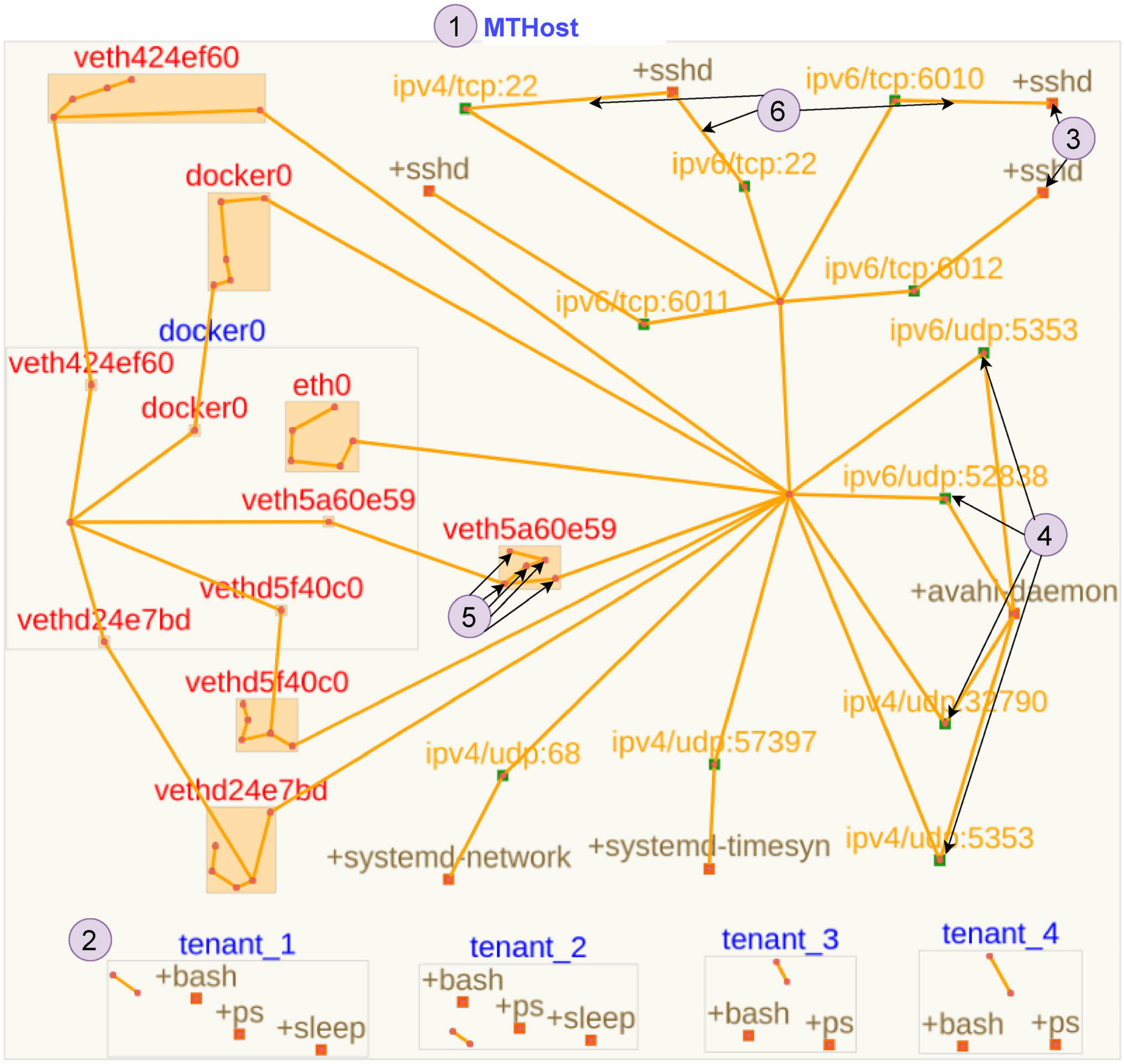


Fig. 2 Discovered service topology example using the proposed ServiceNet with 1 Multi-Tenant Host and 4 tenants

components that were physical in a new virtualised environment. Those virtualised environments can be executed and managed by a multi-tenant host.

Each tenant's data is isolated from the other tenants even when the share the application instance, being even invisible. This ensures data security and privacy for all tenants. Here is where visualising the topology plays a vital role. Due to the growing technology times where everything tends to be softwarised, it can become impossible to maintain a registry and a complete comprehension of the services running, as well as their connection to the network.

Each tenant will publish the updates and they will be treated with the same priority because there is currently no difference between them. It will update the new resources received equally.

The approach presented in this work is not limited by a number in multi-tenant discovery. It can discover the topology of any tenant host available in the system. In Fig. 3 the topology of two tenant hosts is depicted, exemplifying the capability to handle structures of multi-tenant environments.

In this architecture, a topic-based distribution model is used to perform multiple message routes between publishers and subscribers. Each topic represents a specific subject area, such as services topology or device socket

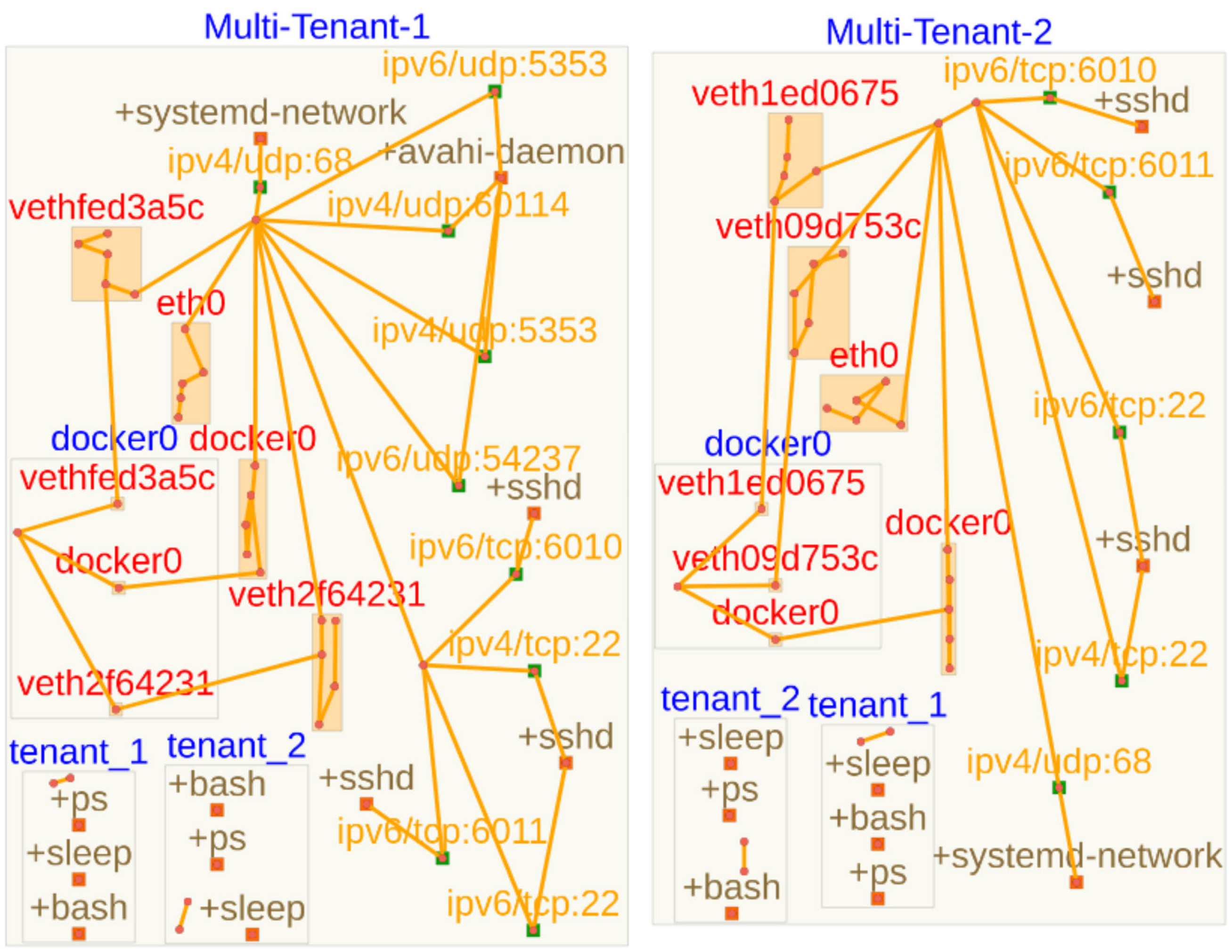


Fig. 3 Discovered service topology example using the proposed ServiceNet with 2 Multi-Tenant Host and 2 tenants each

topology. The messages are published to a topic by a publisher and then delivered to all interested subscribers.

To ensure that messages are delivered accurately it is utilised an intermediary message broker between publishers and subscribers. The broker receives messages from publishers and distributes them to those subscribed to the relevant topics. It also handles message queuing and ensures that messages are delivered in the same order that they were published.

The message broker is based on Advanced Message Queuing Protocol (AMQP) [32]. This protocol provides a standardised messaging format and ensures interoperability between different message brokers and client applications. It enables communication between nodes, addressing the complexities inherent in multi-tenant environments that involve distributed systems. It has been chosen due to its high scalability and fault-tolerance, enabling us to handle large volumes of messages and ensuring that message delivery is not affected by system failures.

In order to make the choice of AMQP, it has been meticulously evaluated against other alternatives, including MQTT [33], XMPP [34], and STOMP [35]. It ensures reliable data delivery and system resilience. In contrast, the MQTT lightweight publish-subscribe structure lacks scalability and security, offering limited features, relying on underlying network security. Unlike AMQP, MQTT supports only a single messaging scenario: publisher/subscriber [36], which may be a problem in multi-tenant architectures. The fact that XMPP focus on real-time communication, hinders its suitability for message reliability and scalability requirements. Similarly, STOMP simplicity comes at the expense of advanced features and scalability, making AMQP the superior choice for managing and visualising complex interactions within the multi-tenant system.

The *TD registry* component is in charge of receiving all the topology information published in the middleware. It is subscribed to each one of the Exchanges created for each of the tenants and information channel. There are currently two different queues for each tenant: network topology and service topology. To maintain the separation of data, a consistent routing key convention is employed, using the format “services.tda.tenantIP.” In this convention, if the tenant IP is, for instance, “192.168.121.2”, the routing key will be “services.tda.192.168.121.2”. This meticulous organisation ensures that each tenant’s information is securely processed and stored.

Each tenant in our system has a unique message bash. Additionally, each device has a hash associated with the tenant ID, and this system is used to filter information effectively. Each service and device socket also include a field to identify the tenant ID to which they belong. This ensures that information is properly filtered for each tenant, avoiding the possibility of inadvertently discovering services in different tenants with the same name and IP.

TD API gets the topology information and translates it to a visual representation using a graph algorithm. This visual representation is invaluable for understanding the complex interactions within the multi-tenant system.

TD GUI shows the topology of the network and services in a visual way. Services within the system are depicted in a distinctive red, each labeled with its associated definition name. Additionally, sockets, the communication endpoints for services, are depicted in vibrant green, complete with labels displaying their protocol type and port. It offers a comprehensive and user-friendly view, enhancing the overall comprehension and monitoring of the multi-tenant system.

4 Design and implementation

4.1 Design approach

Figure 1 illustrates the architectural design of how is ServiceNet composed. The key component for extracting topology information is the TDA (Topology Discovery Agent). The TDA periodically extracts both service and interface topologies from the multi-tenant infrastructure. The extraction frequency is fully configurable and is determined by the configuration parameters explained in Sect. 4.5. Subsequently, it transmits this data to the TD Registry (Service Topology Discovery Registry) through a middleware that manages how the information is shared. The TD Registry serves as a central repository for this topological data. It receives and stores the data making it accessible to the API whenever needed. The GUI is responsible for rendering this topology for end-users, translating text to images in order to visually show the information.

4.2 Discovery approach

The visualisation presented in Fig. 2 encapsulates the key components and their interactions, offering a general view of ServiceNet.

The multi-tenant system architecture visualisation provides a high-level overview of the discovery approach. In (1) The multi-tenant host is identified. In (2) A tenant is exemplified, illustrating the running services inside of it. The subsequent elements are also found: (3) services, (4) sockets, (5) device ports, and (6) connections. Each of them are properly defined in 4.3.

4.3 Data model

ServiceNet defines a data model to represent topological information within the infrastructure. The model comprises several key concepts, each with a set of attributes. Some of the primary concepts include:

1. Device: This concept is used to define physical or logical devices within the infrastructure. It has attributes such as HostName (a unique identifier for a device) and tenantId (an optional attribute that refers to the owner of the device, if applicable).
2. DevicePort: DevicePort represents any network port within a Device. It includes attributes like interfaceName (the name or label of the network port), hostName (the device where it is located), and tenantId (an optional attribute specifying the device owner, if relevant).
3. Connection: This concept is used to represent connections between Devices, DevicePorts, or between DeviceSockets and Services. It includes attributes such as srcResource (the source resource of the connection), srcResourceId (an identifier for the source resource), dstResource (the destination resource of the connection), and dstResourceId (an identifier for the destination resource).
4. Service: Service represents any software component within the computer system that performs a defined task. It includes attributes like ServiceId (the unique identifier for the service assigned by the kernel when it is created), DeviceId (the identifier for the device hosting the service), and type (the space where the service is executed, user space or kernel space).
5. DeviceSocket: DeviceSocket is used to define each of the Sockets on the system. It includes attributes like DeviceId (the identifier for the Device where the socket is located), Direction (specifies whether the socket is a server or client), Binding address (the network address to which the socket is bound), and Port (the port number associated with the socket).

An example of the concepts of the data model is found in Fig. 4, which shows the different types of Device and Resources found in our topology. As can be seen each component has its own resourceId based on specific values to create a unique identifier. In the figure is depicted a multi-tenant host containing one Tenant, one Service and Device Socket to make the example as simple as possible, but keeping it informative. The multi-tenant host is named “MTHost” and its type is “PHYSICAL_HOST”, its resourceId is used to refer to the device where the resources/tenants are found. It is found inside an orange square. The tenant is represented in a grey box as a Device with deviceType “VIRTUAL HOST”. It is then connected to the Host through DEVICE PORTS that represent the interfaces in the tenant and in the host, finally a software switch acts as a bridge created by docker0. The Service in the MThost is connected to the Device Socket due to their ServiceId. They have information that define its functionality as Sid (Service ID) and commandLine for Service, and ipAddress and Port for the Device Socket. Regarding the Connection entities, srcResourceId and dstResourceId refers to the resourceId of the source and destination concepts connected.

By defining each concept and their associated attributes ServiceNet provides a structured and organised way to describe the topology of the infrastructure. This makes it easier to identify and understand the elements within the topology.

4.4 Workflow details

To provide a comprehensive insight into the interaction of various architectural components within our proposed framework, we present a sequence diagram in Fig. 5. The software components, TDA, API, and GUI, are instrumental in achieving service topology discovery and visual representation. They use exchanges to communicate by using a Publisher/Subscriber message broker middle-ware. These exchanges are named Interfaces Exchange, Services Exchange and Structure Exchange.

The workflow is organised into multiple loops, each of them representing a key task. The first loop focuses on the gathering and reporting of topological information, with messages being published to either the Interfaces Exchange or Services Exchange based on their nature. It is started by STDA, which delay is fully configurable by the previously commented parameter (TDA_PERIODIC_REPORTING_INTERVAL). These messages are subsequently received by the component that is subscribed to those two exchanges, the API, which forms the second loop. The API component then processes and generates the topology structure, publishing it to the Structure Exchange. The GUI

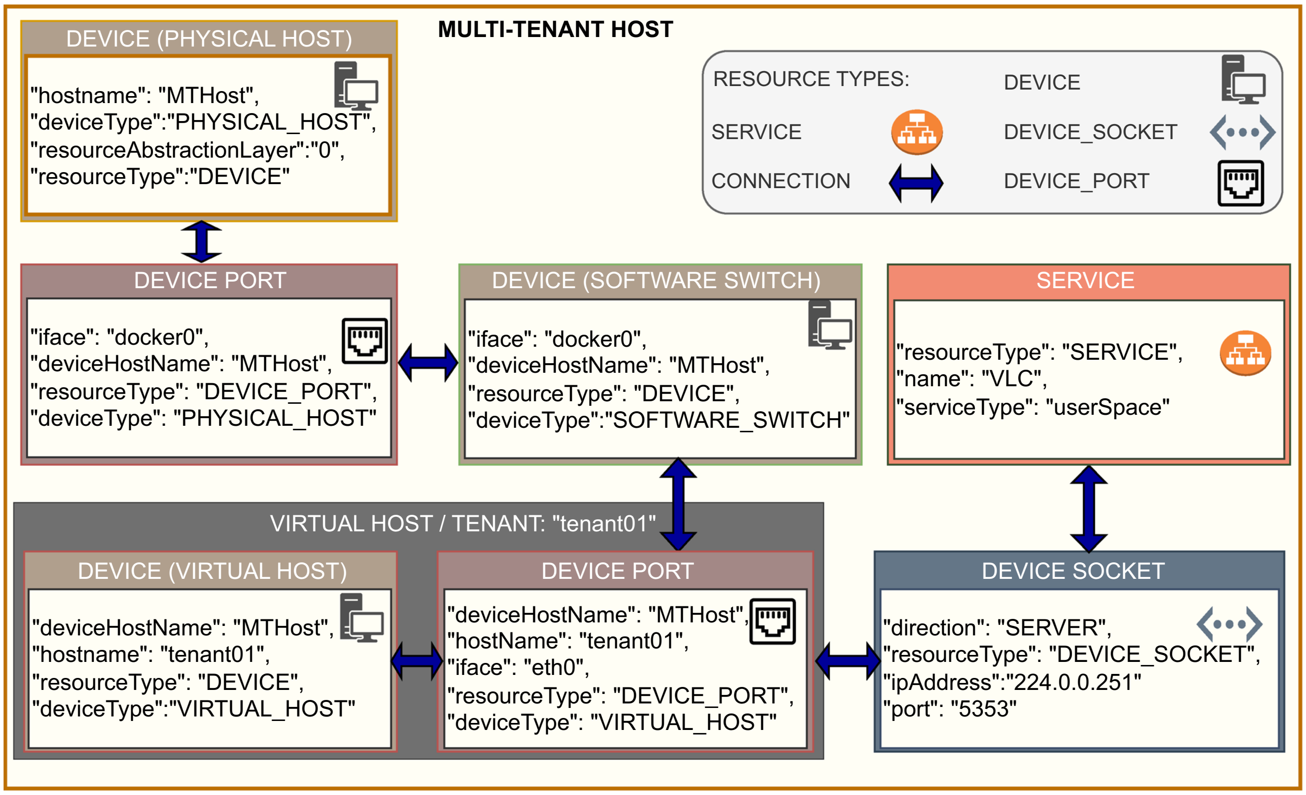

MULTI-TENANT HOST
DEVICE (PHYSICAL HOST)
"hostname": "MTHost",
"deviceType":"PHYSICAL_HOST",
"resourceAbstractionLayer":"0",
"resourceType":"DEVICE"
RESOURCE TYPES:
DEVICE
SERVICE
DEVICE_SOCKET
CONNECTION
DEVICE_PORT
DEVICE PORT
"iface": "docker0",
"deviceHostName": "MTHost",
"resourceType": "DEVICE_PORT",
"deviceType": "PHYSICAL_HOST"
DEVICE (SOFTWARE SWITCH)
"iface": "docker0",
"deviceHostName": "MTHost",
"resourceType": "DEVICE",
"deviceType":"SOFTWARE_SWITCH"
SERVICE
"resourceType": "SERVICE",
"name": "VLC",
"serviceType": "userSpace"
VIRTUAL HOST / TENANT: "tenant01"
DEVICE (VIRTUAL HOST)
"deviceHostName": "MTHost",
"hostname": "tenant01",
"resourceType": "DEVICE",
"deviceType":"VIRTUAL_HOST"
DEVICE PORT
"deviceHostName": "MTHost",
"hostName": "tenant01",
"iface": "eth0",
"resourceType": "DEVICE_PORT",
"deviceType": "VIRTUAL_HOST"
DEVICE SOCKET
"direction": "SERVER",
"resourceType": "DEVICE_SOCKET",
"ipAddress":"224.0.0.251"
"port": "5353"

Fig. 4 Service-Net data model example in a multi-tenant host

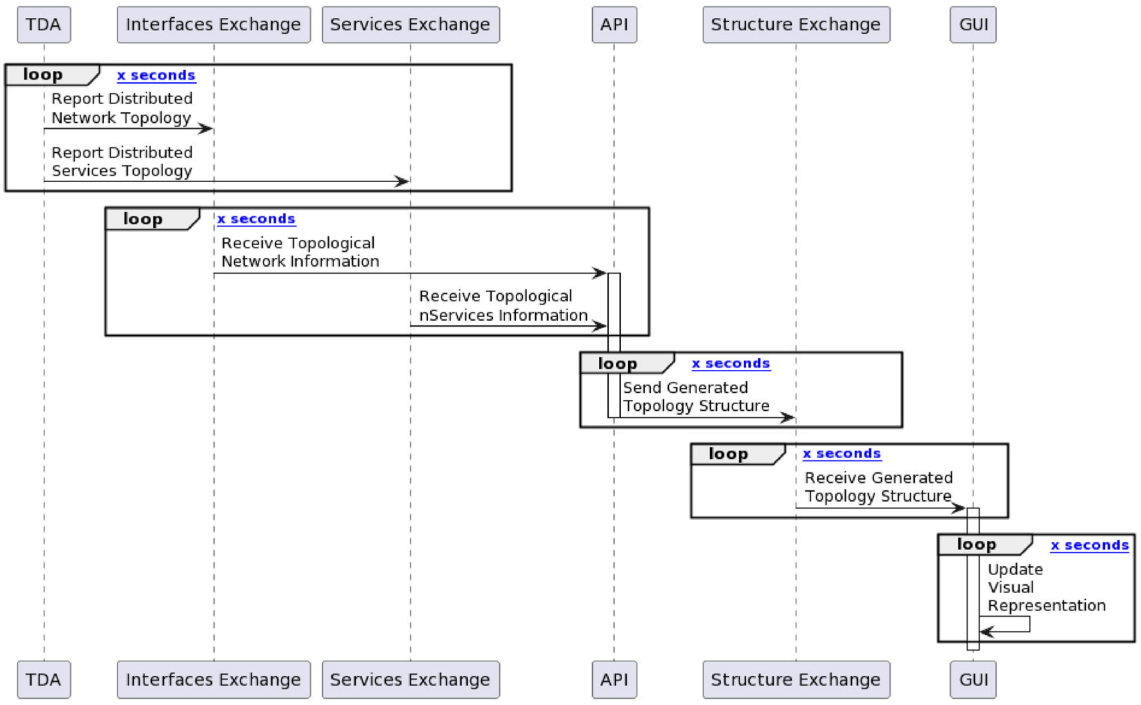

TDA
Interfaces Exchange
Services Exchange
API
Structure Exchange
GUI
loop
x seconds
Report Distributed Network Topology
Report Distributed Services Topology
loop
x seconds
Receive Topological Network Information
Receive Topological nServices Information
loop
x seconds
Send Generated Topology Structure
loop
x seconds
Receive Generated Topology Structure
loop
x seconds
Update Visual Representation

Fig. 5 Workflow in ServiceNet

component is subscribed to this last Exchange, marking the end loop. Finally, the GUI will update the the topology structure rendering to show the new services or network components available in the multi-tenant system.

After the process has finished, it will wait for the delay time to start the discovery again. If it lasted more time than the reporting interval, it will start right after the whole loop has ended. If any error has been encountered, after the value given by TDA_PERIODIC_FORCE_CHANGE_-INTERVAL, it will restart the loop, ensuring error tolerance of our proposed approach.

4.5 Implementations details

Regarding the implementations details, the experiments are executed in a Linux environment. To streamline our resource usage and isolate experiments, we utilise Virtual Machines. Vagrant is used to build and manage the VM with libvirt as the underlying technology, a choice driven by its open-source nature, performance, and scalability, ensuring our experiments run optimally. Each of the tenants is running on docker containers, which leverages kernel containers and cgroups to provide isolation. All the components have been implemented in Java 13. The TDA has been integrated with Linux namespaces/containers (ip netns), Linux inventory tools to gather both service topology (ps, ss, conntrack) and devices topology (lshw and lspci). JavaScript is used to interact with Cytoscape, allowing for the creation of dynamic and interactive network graphs in our GUI [37]. Furthermore, the message broker in charge of the data communication is RabbitMQ version 3.12.7, it is based on Advanced Message Queuing Protocol (AMQP). To disrupt the discovery process by flooding the system with update events, an adversary would require authorised access, including valid credentials and permissions within the system. It has been chosen due to its high scalability and fault-tolerance, enabling us to handle large volumes of messages and ensuring that message delivery is not affected by system failures. All data is stored and accessed through a relational database system, MySQL, ensuring data integrity and efficient retrieval.

Regarding TDA component, it has different configure options in order to provide a complete set of adaptability depending on the desired discovery in each scenario. For instance, you can adjust the following service and socket configuration parameters based on your needs: TDA_REPORT_SERVICE_TO_DEVICE_SOCKET_CONNECTION

TDA_REPORT_DEVICE_SOCKET_CONNECTION

TDA_REPORT_DEVICE_TO_DEVICE_SOCKET_CONNECTION

TDA_REPORT_DEVICE_SOCKETS=true/false

TDA_REPORT_ONLY_DEVICE_SOCK-ETS_WITH_SERVICE TDA_REPORT_ONLY_CHANGES

```
  TDA_REPORT_TENANT_
TOPOLOGY
  TDA_REPORT_ONLY_SERVICE_WITH_DEVICE_
SOCKET
  TDA_PERIODIC_REPORTING_INTERVAL=n1
  TDA_PERIODIC_FORCE_CHANGE_INTERVAL=n2
  TDA_RESOURCE_EXPIRATION_TIME=n3
```

All configure parameters are boolean, taking on true or false values, with the exception of the last three, which are integers representing time intervals. TDA_PERIODIC_REPORTING_INTERVAL sets the delay to start the topology gathering. TDA_PERIODIC_FORCE_CHANGE_INTERVAL is designed to trigger a fresh topology gathering resources. This ensures that the system initiates a new discovery loop either in the absence of errors that might have halted the previous loop, or upon detecting dead or terminated resources. TDA_RESOURCE_EXPIRATION_TIME is used to remove resources from the TD registry. It may occur that a service that is no longer running is still seen in the GUI. TDA addresses this issue by setting the state of the resource as expired when currentTime - resourceTime is higher than TDA_RESOURCE_EXPIRATION_TIME. Subsequently, it will be removed from the GUI. This helps to ensure that the GUI remains up-to-date and does not display any outdated or irrelevant information.

To allow configuration for different scenarios where real-time may not be crucial, there is no minimum time by default. The only requirement is that TDA_PERIODIC_REPORTING_INTERVAL should be lower than TDA_PERIODIC_FORCE_CHANGE_INTERVAL. Failure to meet this requirement will result in the forceful initiation of a new gathering process before its scheduled time.

5 Validation and empirical results

In this section, we provide an empirical evaluation and analysis of the novel architecture for service topology discovery. We focus on key parameters in terms of delivered QoS, including, responsiveness, scaling, number of sockets and number of processes/threads within threads. Furthermore, it is evaluated in various scenarios with different properties.

5.1 Testbed description

The experiments are carried out in a computer with an Intel(R) Xeon(R) CPU E5-2630 v4 @2.20GHz with 20 cores and 32 GB of RAM, running Ubuntu 20.04 LTS as the operating system. To emulate a multi-tenant environment, we employ a VM. The VM is allocated 12 cores and 16 GB of RAM, creating containers inside of it in order to mirror the conditions of multi-tenant infrastructures where diverse tenants coexist and share resources. The aim of doing it in a VM is to have a controlled system, where we can represent best visible scenarios that are easier to show and understand as a starting point.

This testbed allows us to assess the functionality of our service topology discovery framework in a multi-tenant context, where tenants demand efficient resource isolation, scalability, and effective resource management. It is aligned with the real-world scenarios of modern 5G/6G networks, where multiple services and users share common computing resources of a host while having isolation and optimised performance.

5.2 Validation results

In Fig. 2, we present a comprehensive visualisation of a multi-tenant system architecture. This architecture showcases the innovative features of our service topology discovery framework and their potential implications for future industries, particularly in the context of 5G/6G networks.

5.2.1 Graph algorithm

The visualisation of the multi-tenant system architecture is generated using the open source graph library Cytoscape [37], as commented previously in Sect. 4.5. It processes data from the tables stored in our MySQL database and transforms it into a graph representation. The graph visualisation uses the Cola layout with compound node support, providing an organised and visually comprehensible view of the network topology, services, and their interconnections. It also allows for different extensive configurability, allowing gestures such as grab and drag, zoom or multiple selection and also options for customising the arrangement of nodes and edges.

5.2.2 Color key

To enhance clarity, we have adopted a color-coding convention. Host and tenant names are highlighted in blue. Services are represented in red, each labeled with its respective service name, displayed on top of the red square. They are denoted as "+service name". Device sockets are depicted in green, and they provide critical information about their communication. Each socket is labeled with the internet protocol, followed by "/", then the transport protocol (TCP/UDP), and finally, the port number. This information is displayed on top of the green square and is presented as "protocol/TCP-UDP/port". The connections between resources are depicted with orange lines. The interfaces are shown with a big orange square and its name with red on top such as “eth0” and “veth5a60e59”. Conceptually in our data model, each of the points in orange inside them is referred to as device ports, which represent network technologies used to monitor and control the network.

5.2.3 Service and socket discovery

The graph features seven distinct services, each connected to their respective sockets within the host, which are a total of eleven. This outcome is attributable to the configuration setting of the parameter:

```
  TDA_REPORT_ONLY_SERVICE_WITH_DEVICE_
SOCKET
```

Which is set to true disabling the rest of services that do not have sockets. With this parameter we ensure that although all the services are saved in the database, for a matter of clarity we only show those connections. This innovative approach to service and socket discovery is a significant feature of our framework.

5.2.4 Multi-tenant environment

The architecture is not limited to a single tenant but also for and unlimited number of tenants, in the figure shown: `tenant_1`, `tenant_2`, `tenant_3`, and `tenant_4`. They are defined as “DEVICE” with type “VIRTUAL_HOST” (See Fig. 4). Within this multi-tenant context, each tenant hosts between 2 to 3 services. This showcases the framework adaptability and scalability to meet the demands of a diverse multi-tenant environment. The depicted architecture proposes a significant innovation with profound implications for various industries, particularly those at the forefront of 5G/6G network development. As industries continue to face the challenges and embrace the opportunities of the 5G/6G era, the insights and capabilities offered by our service topology discovery framework stand as a powerful resource for future innovations and applications by giving ability to efficiently and comprehensively discover and visualise the complex interplay between services, sockets, and devices.

5.3 Base line results

The testing environment makes use of a minimal host configurations intended to show a reference base line where there are no tenant available. This results can be used as a base line for comparison to investigate overheads relatated to tenant management.

For the gathering of percentage of CPU and memory heap, it employed the widely recognised tool Visual VM [38] (see Fig. 7). This software is a useful resource for observing Garbage Collector (GC) patterns and it is included with the Java Development Kit (JDK). It aids in comprehending memory management and system behavior during the experiments. Although CPU usage values are then represented with our own code in order to enhance visibility.

This experiment is the simplest scenario, we explore the baseline performance. There are no tenants started or running in the multi-tenant system. It provides valuable insights into the framework’s minimal resource consumption. It is worth to note the basis of our approach and then compare it to more complex experiments. In Figs. 6 and 7 can be seen %CPU and heap memory behaviour during a execution. The agent consumes an average of 0.85% of CPU with a peak of 4.3%, the heap memory used has a constant value of 250 MB. Being the heap memory the specific portion of the computer memory where dynamically allocated memory is used, for example, to store data and objects created during the execution of the computer program, in this case, the ST agent. Also, referring to each of the task that the ST agent does regarding service and their sockets discovery, which is one of the main cornerstones of our approach, in Fig. 8 are shown different metrics of this experiment. Regarding the average values for each task, the sockets gathering is 43.9 ms, services gathering 30.2 ms, and the publication of both resources using the topic-based protocol has been 7.5 ms, The total elapsed time for the full discovery is 94.9 ms, completing the discovery in around 10.54 times per second. These metrics and observations in the simplest scenario lay the groundwork for the next experiment, enabling a comparative analysis of the performance.

5.4 Most overloaded scenario (40 tenants)

This experiment investigates the performance of the system with a higher number of tenants, providing a first insight into its scalability previous to the Sect. 5.5. In this section the topology is being collected for 40 tenants. A comprehensive view of the experiment results of the key performance metrics is shown in Fig. 11.

The percentage of CPU utilisation is closely examined in Fig. 9, noticing minor increase when compared to the experiment with minimal configuration. the agent consumes an average of 1.21%, with a peak value of 5.10%, reflecting the higher workload due to the additional tenants, but not a big change probably due to the fact that now there are more tenants needing resources to work. Additionally, the heap memory behavior, illustrated in Fig. 10, remains

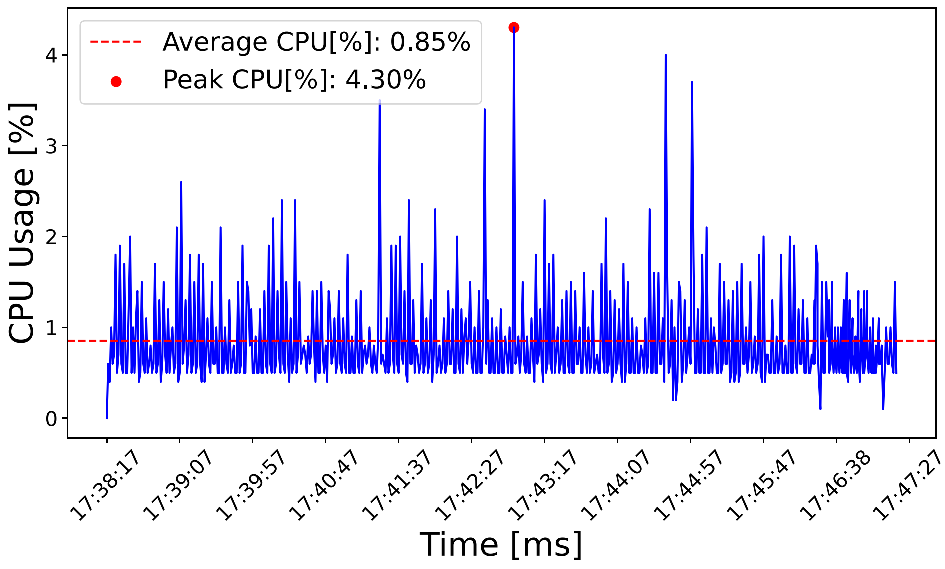


Fig. 6 CPU usage in base line experiment

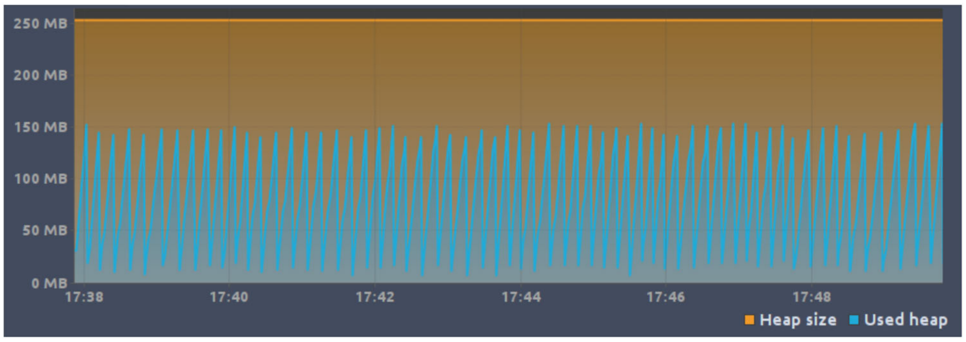


Fig. 7 Reserved and used heap memory in base line experiment

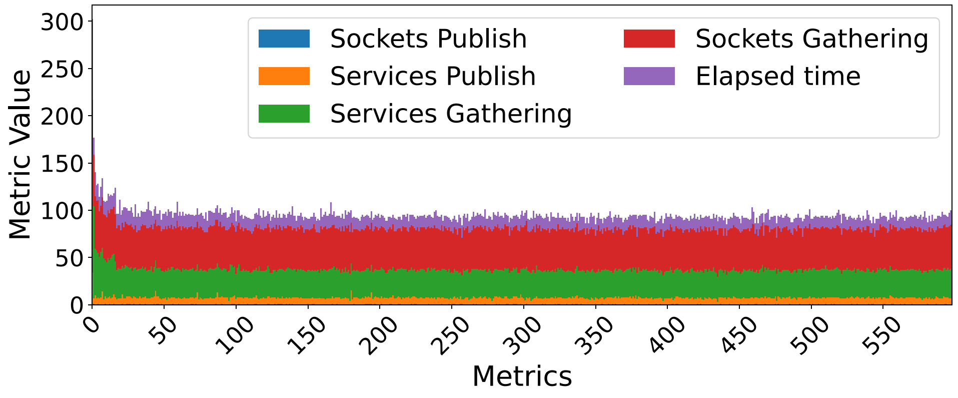


Fig. 8 Time spent in each task in base line experiment

consistent with the one seen in the Base Line, using a constant heap memory of 250 MB.

This experiment also assesses the different tasks done by the ST agent. As seen in Fig. 11, tenant services gathering takes 633.1 ms per average, sockets gathering consumes 64.5 ms, services gathering lasts 40.6 ms, and the publication of both resources using a topic-based protocol completes in 7.8 ms. The total elapsed time for the full discovery is 806.4 ms. This demonstrates that while the agent requires more time for completing topology gathering, it is still able to manage higher tenant workloads.

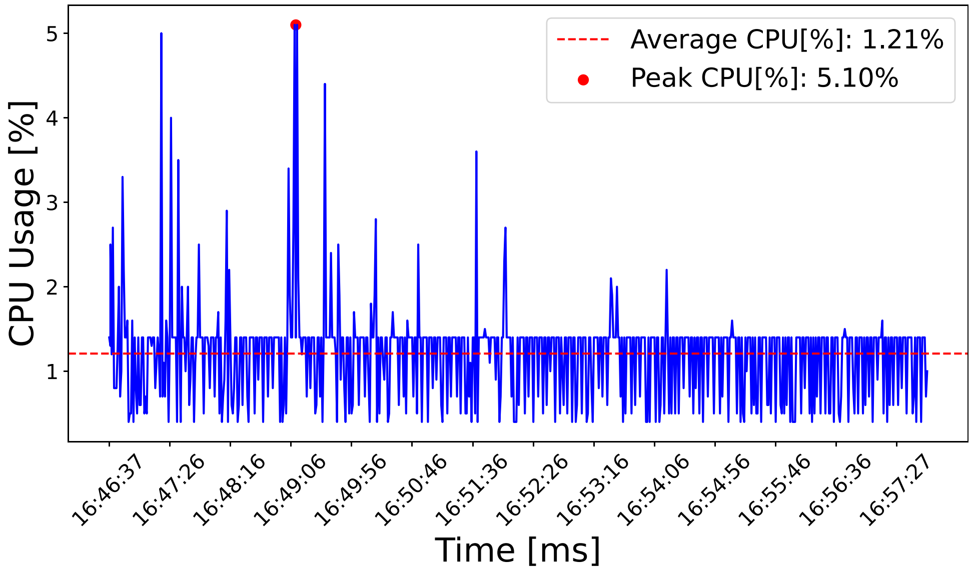


Fig. 9 CPU usage in most overloaded experiment

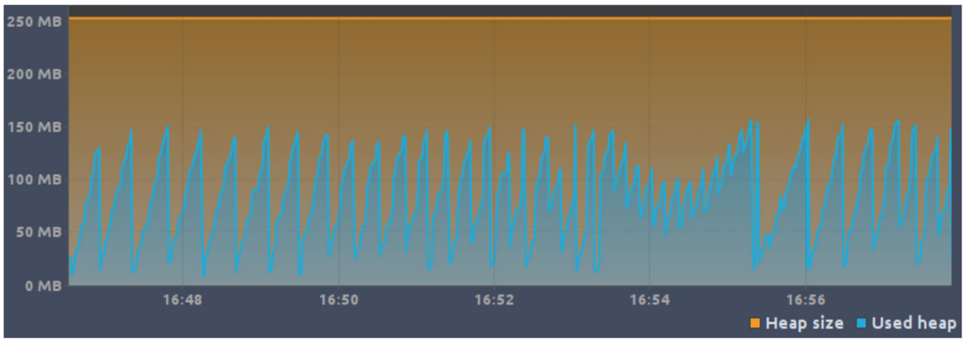


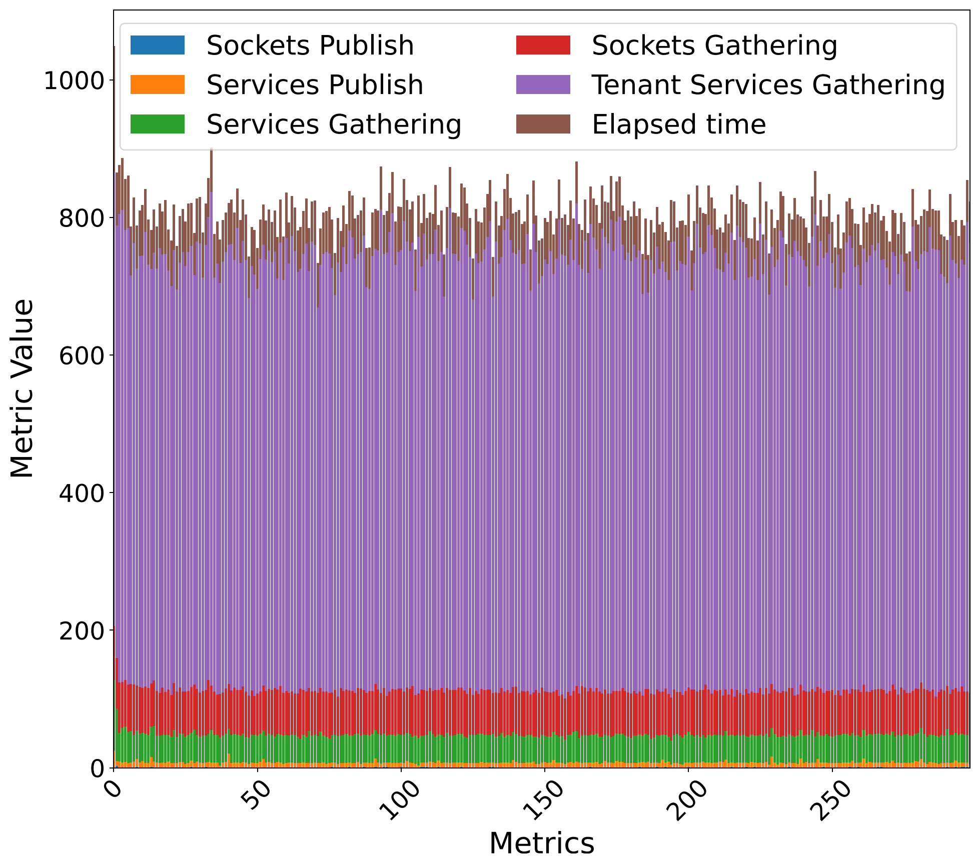
Fig. 10 Reserved and used heap memory in most overloaded experiment

5.5 Fine-grain analysis of tenant scalability

In this subsection, we examine the scalability of our system, particularly in handling a large-scale distributed architecture across varying numbers of tenants. The metrics presented in Fig. 12 provide insights of the performance depending on the tenants that are running on the system. The number of tenants is systematically varied from five to forty, allowing us to assess the performance of the system across a range of scenarios. The following metrics are gathered: Total elapsed time, Tenant Services Gathering, Services Gathering, Sockets Gathering, Services Publish and Sockets Publish. Those metrics reveal the distributed ability to efficiently manage diverse workloads and maintain responsiveness across the different scenarios.

The total elapsed time is an essential factor in the analysis of scalability, being calculated as the time since the loop starts until it makes a complete gathering and reporting. We can observe how it tends to increase as the number of tenant grows. This is expected since a larger tenant base introduces additional workload to the system, not only exposing the ST agent to higher workload, but also consuming more system resources. The tenant services gathering process also demonstrates scalability as the number of tenants grows, indicating a reasonable scaling trend. It is relatively normal because of the same reason as the total elapsed time increase. The minor changes observed in multi-tenant services gathering and sockets gathering can be due to several factors, including the allocation of system resources and the scalability of these specific tasks. For example, multi-tenant services gathering (depicted in green) varies from 38 ms with five tenants to 46 ms with the maximum number of tenants. A similar pattern is observed in sockets gathering (depicted in red), which takes 53 ms with five tenants, 56 ms with ten tenants, and 61 ms with fifteen tenants, and it then reaches a maximum of 76 ms with forty tenants, showing minimal changes. The stability in these metrics may be influenced by how system resources are shared and utilised across different tenant scenarios. This is also seen in Fig. 9 where the percentage of CPU does not notice a big change in the load even in the forty tenant case, possibly due to that they are also consuming and limiting resources. The last two metrics are the publication of services and sockets. The first one, in purple, indicates a variation of only 41 s from the minimum

Fig. 11 Time spent in each task in most overloaded experiment



to the maximum of Services Publish elapsed time, being the values 7 ms and 8 ms. And finally, the Sockets Publish, is always 1 ms, remarking that it is the fastest task in every case.

Regarding memory scalability, it has been seen a “Healthy Saw-Tooth Pattern” in heap memory utilisation (Figs. 7, 10). This pattern is indicative of application health as commented in [39]. It suggests that garbage collection processes are functioning correctly. The saw-tooth pattern reflects consistent rises in heap usage, followed by ‘Full GC’ events, after which memory utilisation sharply drops. This cyclic pattern, reaffirms the robustness of the system, indicating that is is table and manages memory effectively.

Notably, even in the most challenging case with 40 tenants, the total elapsed time remains around 892 ms, demonstrating highly satisfactory real-time monitoring capabilities for service topology discovery.

5.6 Discovery and update convergence

Understanding how ServiceNet converges during the initial discovery or when an update is crucial for evaluating its performance. The study of Elapsed Time (ET) in Table 3 provides insights into the convergence dynamics under different scenarios.

Scenarios are divided by the number of Multi-Tenant systems (M-T), the number of Tenants (T) and the different stages in the discovery lifecycle. The stages are from the initial discovery (Cold Start), to the moment when an update has occurred Hot Update, and finally Forced Update, which is happening when a resource has not been updated in a given time. Forced Update is needed to ensure absence of errors or detect terminated resources that should be removed from the TD Registry.

The average ET shows time the since the resource is gathered by the TDA until it is inserted in the TD registry. As the number of M-T and T increases, there is a general upward trend in the ET, suggesting that the system takes more time to handle a larger number of both multi-tenant and tenant. For scenarios with M-T = 2 and T>0 Cold Start and Forced Update show comparable ET, while Hot Update typically takes less time, as expected due to updating existing information. The increasing time may indicate potential system resource limitations. High values, specially in scenarios with more Tenants, may imply that the system is approaching or exceeding its processing capabilities. An example can be seen in the last scenario, resulting in 200 tenants being executed at the same time in a single computer (see computer specifications in 5.1).

Fig. 12 Elapsed time depending on the number of tenants

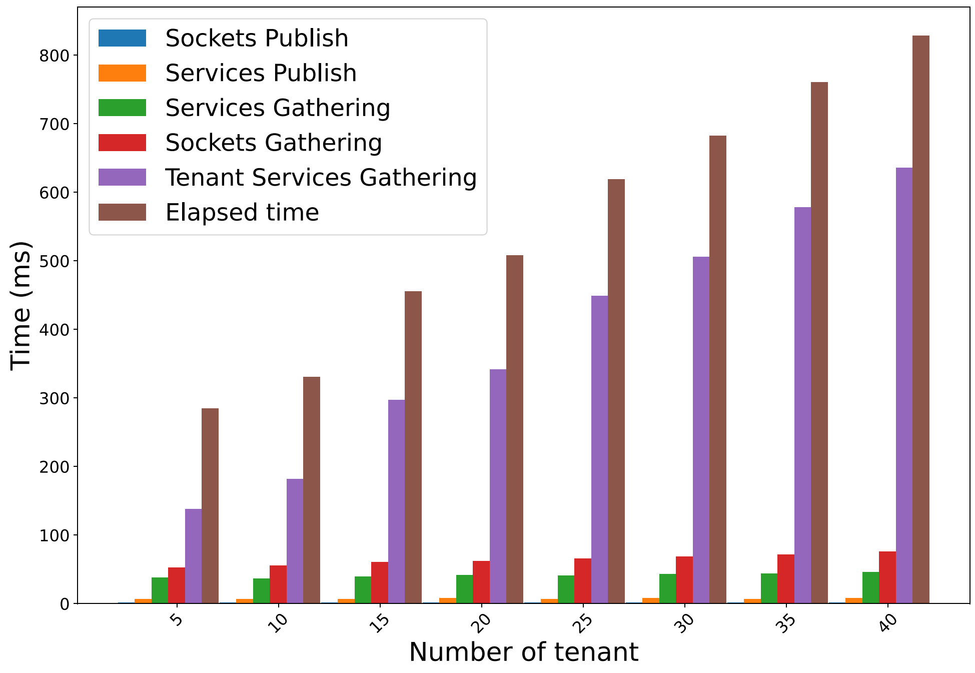


Table 3 Average elapsed time (ET) from TDA to TD registry insertion (in s) for Services and Device Sockets

Scenario		Services			Device sockets		
M-T	T#	Cold start	Hot update	Forced update	Cold start	Hot update	Forced update
1	0	0.828	0.688	0.696	0.653	0.603	0.595
2	0	0.855	0.688	0.710	0.995	0.774	0.944
2	20	1.378	1.527	1.507	0.943	0.966	0.894
2	40	1.601	1.948	1.876	0.947	0.967	0.894
2	60	1.908	2.358	2.270	0.923	0.961	1.206
2	80	2.169	2.748	2.636	0.944	0.968	0.894
2	100	2.486	3.216	3.062	0.934	0.966	0.895

(#)T: number of tenants for each Multi-Tenant (M-T)

6 Conclusion

In this paper, a novel architecture has been designed, prototyped and validated to meet the challenging requirements of 5G and beyond networks, where deep understanding of network topologies, services, and their interconnections is essential for efficient resource management in large-scale environments. The proposed ServiceNet architecture is widely applicable and useful to discover complex topologies within multi-tenant systems and virtualised environments. Furthermore, a new solution based on a novel Topology Discovery Agent has been introduced to efficiently discover both network and service information, seeking to allow complete comprehension of the big picture. The adaptability and configurability of the solution allow it to be fine-tuned controlled to discover services depending on the specific requirements. An accurate and efficient data transmission is ensured due to the use of a message-oriented middleware and a topic-based distribution model. Moreover, the use of the Service Topology Discover Registry ensures secure data storage, crucial in dynamic networking environments. A realistic testbed has been deployed to test, validate and evaluate the proposed approach. Empirical results have demonstrated the high scalability of the approach over different number of tenants, as well as different metrics to understand how the system performs, in spite of the limited resources of the testbed. The CPU usage is low in both experiments with an average of 0.85% and 1.21%, this allows the agent to be non-invasive. The time to finish each loop goes from 285ms to a maximum of 829ms in the worst scenario, which shows a satisfactory time for topology discovery even in challenging scenarios. Lastly, the visualisation of a large-scale multi-tenant system has been presented, providing a valuable tool for complete understanding, representing the data gathered using the data model created with different resources and devices.

In the realm of future work, this paper opens up a wide variety of opportunities. First, delving into the incorporation of advanced security mechanisms and encryption protocols would fortify the TD registry. Moreover, continuous refinement and expansion of the visualisation tools for large-scale multi-tenant systems could contribute to a more comprehensive topology understanding. A good example could be using virtual reality to allow for a 3D view of the whole system and its interconnections. Lastly, the integration with orchestration tools such as OpenShift and Kubernetes is the next planned step, as it is recognised by the authors that it would provide broader applicability.

Acknowledgements This work is in part funded by the European Commission Horizon 2020 5G-PPP Program under Grant Agreement Number: H2020-ICT-2020-2 / 101017226 "6G BRAINS: Bringing Reinforcement learning Into Radio Light Network for Massive Connections".

Author contributions Experiments, figures, and main manuscript text performed by the corresponding author (Angel M. Gama). Conceptualization, methodology, and review conducted by the supervisors (Jose M. Alcaraz, Higinio Mora, and Qi Wang).

Funding European Commission Horizon 2020 5G-PPP Program under Grant Agreement Number: H2020-ICT-2020-2 / 101017226 "6G BRAINS: Bringing Reinforcement learning Into Radio Light Network for Massive Connections".

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

Declarations

Competing interests The authors have not disclosed any competing interests.

References

1. Deng, X., Wang, L., Gui, J., Jiang, P., Chen, X., Zeng, F., Wan, S.: A review of 6g autonomous intelligent transportation systems: mechanisms, applications and challenges. J. Syst. Archit. **142**, 102929 (2023). https://doi.org/10.1016/j.sysarc.2023.102929
2. Ghildiyal, Y., Singh, R., Alkhayyat, A., Gehlot, A., Malik, P., Sharma, R., Akram, S.V., Alkwai, L.M.: An imperative role of 6g communication with perspective of industry 4.0: challenges and research directions. Sustain. Energy Technol. Assess. **56**, 103047 (2023). https://doi.org/10.1016/j.seta.2023.103047
3. Nzanywayingoma, F., Yang, Y.: Efficient resource management techniques in cloud computing environment: a review and discussion. Int. J. Comput. Appl. **41**(3), 165–182 (2019). https://doi.org/10.1080/1206212X.2017.1416558
4. Chirivella-Perez, E., Salva-Garcia, P., Sanchez-Navarro, I., Alcaraz-Calero, J.M., Wang, Q.: E2e network slice management framework for 5G multi-tenant networks. J. Commun. Netw. **25**(3), 392–404 (2023). https://doi.org/10.23919/JCN.2023.000019
5. Garcia, A.M.G., Calero, J.M.A., Mora, H.M., Wang, Q.: Process slicing: a new mitigation tool for cyber-attacks against softwarised industrial environments. In: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), pp. 468–473. (2023). https://doi.org/10.1109/NetSoft57336.2023.10175447
6. Baldin, I., Ruth, P., Wang, C., Chase, J.S.: The future of multiclouds: a survey of essential architectural elements. Int. Sci. Tech. Conf. Mod. Comput. Netw. Technol. **2018**, 1–13 (2018). https://doi.org/10.1109/MoNeTeC.2018.8572139
7. Sanchez-Navarro, I., Bernal Bernabe, J., Alcaraz-Calero, J.M., Wang, Q.: Advanced spatial network metrics for cognitive management of 5G networks. Soft Comput. **25**(1), 215–232 (2021)
8. Groba, C., Clarke, S.: Synchronising service compositions in dynamic ad hoc environments. IEEE First Int. Conf. Mob. Serv. **2012**, 56–63 (2012). https://doi.org/10.1109/MobServ.2012.16
9. Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., Veltri, L.: A scalable and self-configuring architecture for service discovery in the internet of things. IEEE Internet Things J. **1**(5), 508–521 (2014). https://doi.org/10.1109/JIOT.2014.2358296
10. Mathews, D.R., Verma, M., Lakshmi, J., Aggarwal, P.: Towards more effective and explainable fault management using cross-layer service topology. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), pp. 94–96. (2022). https://doi.org/10.1109/CLOUD55607.2022.00026
11. SolarWinds, Network topology mapper. (2023). https://www.solarwinds.com/network-topology-mapper
12. NMSaaS, Comprehensive network and configuration management for service providers and enterprise organizations. (2023). https://www.nmsaas.com/
13. Paessler, Prtg network monitor. (2023) https://www.paessler.com/
14. Wei, H., Rodriguez, J.S., Garcia, O.N.-T.: Deployment management and topology discovery of microservice applications in the multicloud environment. J. Grid Comput. **19**(1), 1 (2021). https://doi.org/10.1007/s10723-021-09539-1
15. Sanchez-Navarro, I., Mamolar, A.S., Wang, Q., Alcaraz Calero, J.M.: 5gtoponet: real-time topology discovery and management on 5G multi-tenant networks. Future Gener. Comput. Syst. **114**, 435–447 (2021). https://doi.org/10.1016/j.future.2020.08.025
16. Juju: The simplest way to deploy and maintain applications in the cloud. (2023). https://juju.is/
17. Capistrano: A remote server automation and deployment tool written in ruby. (2023) https://capistranorb.com/
18. Scalr: American cloud computing company specializing in automation and collaboration software for terraform. (2023) https://www.scalr.com/
19. Perforce: Puppet infrastructure & it automation at scale | puppet by perforce. (2023) https://www.puppet.com/
20. Oracle: Java se remote method invocation apis and developer guides. (2023) https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html
21. Group, O.M.: Corba | object management group. (2023) https://www.corba.org/

22. OpenStack: Open source cloud computing infrastructure-openstack. (2023) https://www.openstack.org/
23. Trainer, T.: Sdp service discovery protocol. (2023). https://www.telecomtrainer.com/sdp-service-discovery-protocol/ Accessed 12 Oct 2023
24. Universitat Rostock: Service discovery protocol (sdp). https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/sdp.html Accessed 08 Jan 2024
25. DNS-SD.org Project, Dns service discovery (dns-sd). http://www.dns-sd.org/ Accessed 08 Jan 2024
26. Avahi.org, avahi - mdns/dns-sd. https://www.avahi.org Accessed 09 Jan 2024
27. Apple Inc., Bonjour-apple developer. https://developer.apple.com/bonjour/ Accessed 09 Jan 2024
28. Duan, Q., Lu, Y.: Service-oriented network discovery and selection in virtualization-based mobile internet. J. Comput. Inf. Syst. **53**(3), 38–46 (2013). https://doi.org/10.1080/08874417.2013.11645630
29. Gephi, Open graph viz platform. https://gephi.org/ (2023)
30. Cummins, H.: Ibm garage methodology: the principles of modern service management. https://www.ibm.com/garage/method/practices/manage/principles-of-modern-service-management/ (2023)
31. Averdunk, I.: Ibm garage methodology: the 5 principles of cloud service management and operations. https://www.ibm.com/garage/method/practices/manage/service-management-five-principles/ (2023)
32. AMQP Open, Amqp | the advanced message queuing protocol. https://www.amqp.org/Accessed 08 Jan 2024
33. OASIS, Mqtt - the standard for IoT messaging. https://www.mqtt.org/ Accessed 08 Jan 2024
34. XMPP Standards Foundation, Xmpp | the universal messaging standard. https://xmpp.org/ Accessed 08 Jan 2024
35. STOMP.io, Stomp protocol documentation. https://stomp.github.io/stomp-specification-1.2.html Accessed 08 Jan 2024
36. Ashtari, H.: Amqp vs. mqtt: 9 key differences. https://www.spiceworks.com/article/274854 Accessed 08 Jan 2024
37. Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T.: Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. **13**(11), 2498–2504 (2003)
38. VisualVM, Visualvm, https://visualvm.github.io/ (2023)
39. Lakshmanan, R.: Interesting garbage collection patterns. https://dzone.com/ARTICLEs/interesting-garbage-collection-patterns (2021)

Angel Gama Garcia is a Robotics Engineer with a Master's Degree in Robotics and Automation. He is currently a Research Assistant and a Ph.D. candidate at the University of the West of Scotland. His research focuses on service management for vertical business use cases, contributing to funded projects such as 6G BRAINS and INCODE, in collaboration with academic and industry partners worldwide.

Jose M. Alcaraz Calero is a Full Professor in networks and security at the University of the West of Scotland, and he is the technical co-coordinator of the EU H2020 5G-PPP Phase I SELFNET and Phase II SliceNet. His professional interests include network cognition, management, security and control, service deployment, automation and orchestration, and 5G mobile networks. He has a PhD in computer Science, University of Murcia, Spain.

Higinio Mora Mora received the first B.S. degree in computer science engineering, the second B.S. degree in business studies, and the Ph.D. degree in computer science from the University of Alicante, Spain, in 1996, 1997, and 2003, respectively. Since 2002, he has been a member of the Faculty of the Computer Technology and Computation Department, University of Alicante, where he is currently an Associate Professor and a Researcher with the Specialized Processors Architecture Laboratory. He has participated in many conferences and most of his work has published in international journals and conferences, with more than 50 published articles. His research interests include computer modeling, computer architectures, high-performance computing, embedded systems, the Internet of Things, and cloud computing paradigm.

Qi Wang is a Full Professor at the University of the West of Scotland, and he is the technical cocoordinator of EU H2020 5G-PPP Phase I SELFNET and Phase II SliceNet. He is a Board Member of the Technology Board of EU 5G-PPP, and Member of USA Video Quality Expert Group (VQEG). His research primarily focuses on 5G mobile networks and video networking. He has a PhD in mobile networking from the University of Plymouth, UK.