

Microservice-Based Fog Testbed for 6G Applications

Ekaterina Kuzmina

Department of Telecommunication
Networks and Data Transmission,
Saint-Petersburg State University of
Telecommunications
Saint Petersburg, Russia
kuzmina120601@yandex.ru

Meriem Tefikova

Department of Telecommunication
Networks and Data Transmission,
Saint-Petersburg State University of
Telecommunications
Saint Petersburg, Russia
tmrvmr@mail.ru

Artem Volkov

Department of Telecommunication
Networks and Data Transmission,
Saint-Petersburg State University of
Telecommunications
Saint Petersburg, Russia
artemnv.work@gmail.com

Ammar Muthanna

Department of Telecommunication
Networks and Data Transmission,
Saint-Petersburg State University of
Telecommunications
Saint Petersburg, Russia
muthanna.asa@spbgut.ru

Abdelhamied A. Ateya

Department of Electronics and
Communications Engineering, Zagazig
University, Zagazig 44519, Egypt;
EIAS Data Science Lab, College of
Computer and Information Sciences,
Prince Sultan University, Riyadh 11586,
Saudi Arabia
aateya@psu.edu.sa

Andrey Koucheryav

Department of Telecommunication
Networks and Data Transmission,
Saint-Petersburg State University of
Telecommunications
Saint Petersburg, Russia
akouch@mail.ru

Abstract— This paper provides a real-time fog computing model based on a microservice architecture that enables testing and modeling of eventual implementations of ultra-reliable low-latency communications (uRLLC) services. The work provides fog-based architecture for sixth-generation cellular (6G) applications, including telepresence and uRLLC. A testbed of a robot swarm was developed to prototype the proposed network architecture. Computing tasks are offloaded and handled based on a proposed microservice scheme introduced to meet the 6G requirements. Furthermore, we developed a novel migration scheme for the proposed architecture to support the mobility of end devices. The optimum server for migrating computing tasks is allocated by solving a proposed optimization problem using particle swarm optimization (PSO). All proposed algorithms were implemented in the developed prototype. The proposed work is introduced to provide an architectural foundation for testing fog-based 6G applications and services and to implement and test novel network methods in the future.

Keywords— fog computing, microservice architecture, 6G networks, telepresence services, uRLLC, Dobot Magician

I. INTRODUCTION

The sixth-generation cellular (6G) system is expected to provide novel services and applications not supported by the previous cellular generations [1]. Such applications include ultra-reliable low-latency (uRLL) applications such as telepresence, holographic, and haptic communications [2]. However, the design of communication networks for 6G applications faces many challenges due to the ultimate requirements of such applications [3]. The 6G systems should achieve ultra-high availability, reliability, and resilience. The end-to-end latency of the communicated data over 6G networks should be sub-millisecond [4]. Developing a testbed for implementing 6G algorithms, methods, and applications is also challenging.

Telepresence services enable users to communicate virtually, even if they may be miles apart, considering using

telepresence technologies [5]. Everyone can establish a real-time sensation of presence and interaction using these services. Technologies like virtual reality, augmented reality, and holography are employed to implement this paradigm [6]. Telepresence can be used in various essential areas, such as remote medical diagnosis, consultations, and industrial automation [7].

The introduction of telepresence services significantly improves the quality of life, optimizes the use of human resources, and meets sustainability requirements. However, designing communication networks that provide telepresence services faces many design challenges, including dependability, reliability, availability, and latency [8]. Telepresence services are ultra-reliable low-latency communication (uRLLC) that require ultra-high data rates and extremely-low latency. Such services are one of the main services of the 6G, as per the third-generation partnership project (3GPP) and the international telecommunication union (ITU) [9, 10].

It is possible to achieve high data transmission rates, high reliability, and fault tolerance for the 6G networks by optimizing particular network segments. Monolithic programs can be swapped out for microservice architectures, and popular cloud computing can be replaced with distributed edge computing (e.g., fog computing) [11]. Microservice architecture, the first element of this optimization, enables extensive monolithic systems to be broken into a collection of

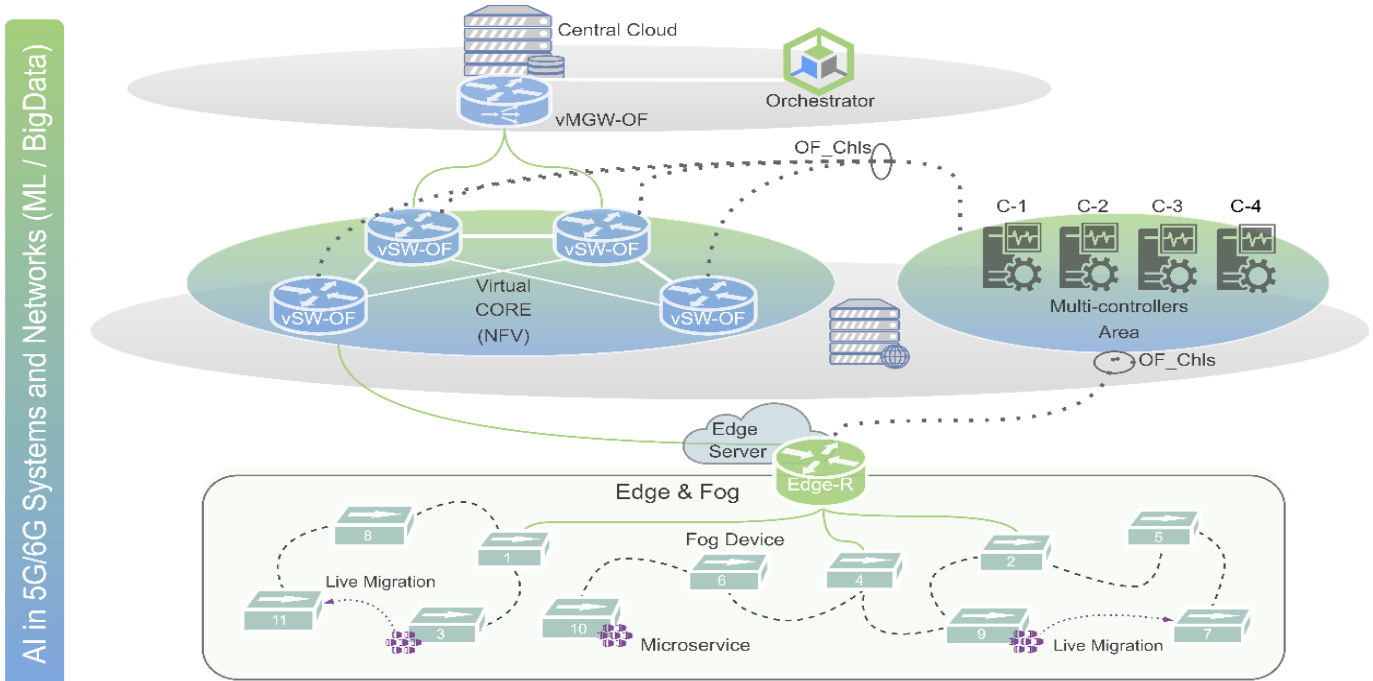


Fig. 1. Fog-based 6G network structure.

interconnected components, known as microservices, which cooperate as a single application [12]. This improves the system's fault tolerance, decreases the deployment complexity, boosts the application's scalability, and speeds up the introduction of new features compared to the conventional approach [13].

While cloud computing always involves data processing only on a remote centralized server, fog computing enables data processing in the end device's domain [14]. This can be achieved by distributing the processing of computing tasks to different fog points through remote micro-servers and user end devices[15]. The data transmission delay can be reduced by processing data closer to the user, hastening the response to a specific user request. Moreover, fog computing enables the creation of "virtual clouds" (also known as computational fog) by creating clouds "out of thin air" [16].

The ultimate objective of this work is to investigate novel strategies for establishing real-time microservice-based fog architecture to implement uRLLC 6G services. The main contributions of this work are summarized as follows.

- Developing a fog-based architecture for 6G applications.
- Developing a testbed of a swarm of robots based on the proposed fog architecture.
- Configuring the developed robots testbed.
- Developing a microservice architecture for the proposed system.
- Implementing the developed microservice scheme on the developed testbed.
- Developing a migration scheme for migrating microservices in the developed system.
- Optimizing the performance of the proposed migration scheme by selecting the optimum server for migration using particle swarm optimization (PSO).

II. FOG-BASED NETWORK STRUCTURE

Fig. 1 presents the proposed fog-based architecture of 6G networks. The higher layer is the core network layer, which deploys a network orchestrator and a central cloud unit of multiple servers with significant computing resources. The network orchestrator is the primary resource management tool, including processing, data storage, bandwidth, and energy.

The core network also deploys the software-defined networking (SDN) paradigm with multiple SDN controllers. SDN is mainly deployed to improve network flexibility, availability, resilience, and security [17]. Network function virtualization (NFV) is deployed over SDN to offer a software-oriented approach to network administration that allows managing network resources through network interfaces [18].

The lower layer of the architecture (Edge & Fog network) is the distributed edge layer that deploys two levels of edge servers: multiple access edge computing (MEC) servers and fog computing servers. Fog nodes interact with edge routers (Edge-R) to transmit data to the higher level of the proposed architecture.

The Edge & Fog network segment nodes are machines that process incoming data through fog computations. Each fog device offers a microservice or a set of them with specific functionality. These devices collectively provide services organized based on a microservice architecture. Microservice migration is also enabled for load balancing, scalability, and application deployment flexibility in the Edge & Fog segment [19]. It is important to pay close attention to the artificial intelligence (AI) block, which permeates every level and component of the design. With the advancements in chip element design and architecture, it is now possible to integrate AI functions for managing network resources [20].

III. PROPOSED TESTBED

We considered a swarm of robots as a use case to implement the previously proposed architecture. Fig. 2 presents the proposed prototype implementation. The Edge & Fog segment introduced in Fig.1 is implemented in the proposed robot implementation.

The proposed testbed consists of four machines (i.e., robots), each consisting of two main parts: moveable and logical.

Moving part: The robotic manipulator is employed to control the stand's dynamics. The deployed robots are configured with the parameters introduced in Table. I.

TABLE I. CONFIGURATION PARAMETERS OF THE MOVING PART OF THE DEPLOYED SWARM OF ROBOTS

Parameter	Value
Working area	340 mm in radius
Number of axes	Four
Accuracy	repeatability up to 0.02 mm placement up to 0.2 mm
Power supply	50/60 Hz 100-240V
Power consumption	200 W

The dynamic segment was built using a flexible 4-axis robotic manipulator deployed in our lab and can carry out a variety of robotics and automation-related tasks. Figure 3 presents the considered robotic arm compatible with many programming languages, including C++, C#, Python, and Java [21].

Logical part (Fog-nodes emulation): This part represents the edge segment implementation. Four Raspberry Pi 4 Model B microcomputers are employed as the stand's logical component to provide edge computing capabilities. The main characteristics of the deployed fog nodes (i.e., Raspberry Pi nodes) are introduced in Table II.



Fig. 3. Robotic arm as an element of the dynamic segment.

TABLE II. CONFIGURATION PARAMETERS OF THE ROBOTS' EMBEDDED FOG NODES

Parameter	Value
Processor	Broadcom BCM2711 Running at 1.5GHz on four ARM Cortex-A72 cores
RAM	LPDDR4-3200 SDRAM (8 GB)
Wireless interfaces	Bluetooth 5.0, BLE, and 802.11ac Wi-Fi
Storage	32 GB MicroSD memory card slot
General-purpose input/output (GPIO)	GPIO header with 40 pins
Power requirements	USB-C or GPIO at DC 5V, 3A



Fig. 2. The developed testbed.

A. Moving part of the Developed Testbed

The movement system of the proposed four robots was implemented using a program code written in Python version 3 that moves the robots over the optimum trajectory along a single course. Thus, the system was built to simulate the motion of a swarm of particles. However, in such architectures, veering off course can frequently happen for many reasons. The construction of the moving part of robots considers any deviation of one of the manipulators from the overall trajectory of the swarm because it is important to tackle several difficulties when deviations occur. Thus, three of the manipulators will continue moving synchronously along a typical trajectory after a certain period (Δt). In contrast, the fourth manipulator will deviate and move randomly to the other three.

The system is developed using client-server architecture, where two personal computers (PCs) connected to the same local network act, respectively, as a client and a server, to enable remote and flexible control of the system. Fig. 4 presents the network diagram of the dynamic segment

PC#1 is connected to the switch as a client, and PC#2 is linked to the switch as a management server over an Ethernet connection. The client PC is connected to four Dobot Magician robots via individual USB 3.0 connectors. Sockets, which are programming interfaces for inter-process communication, are used for communication between the server and the client. Logical addresses and ports are used for the communication between devices utilizing the TCP/IP protocol stack. The Internet protocol version 4.0 (IPv4) and version 6.0 (IPv6) are used in the network layer of the proposed structure. For the IPv4 and IPv6 protocols, the address is a structure of 32 bits and 128 bits, respectively, while the port number is an integer value between 0 and 65535 (for the TCP protocol).

The pair of address and port numbers define the socket. The fundamental benefit of using sockets is that data can be sent and received immediately through the socket. As a result, it is possible to transfer data between two or more devices without getting bogged down in the details of the protocols. The UDP protocol was chosen for data transfer between the server and the client to achieve a high data rate.

We built four client listening sockets on four separate official UDP ports, an external client IP address, and two server sockets that deliver commands to the corresponding client sockets to achieve complete remote control of robots. This architecture enables efficient remote control of robots.

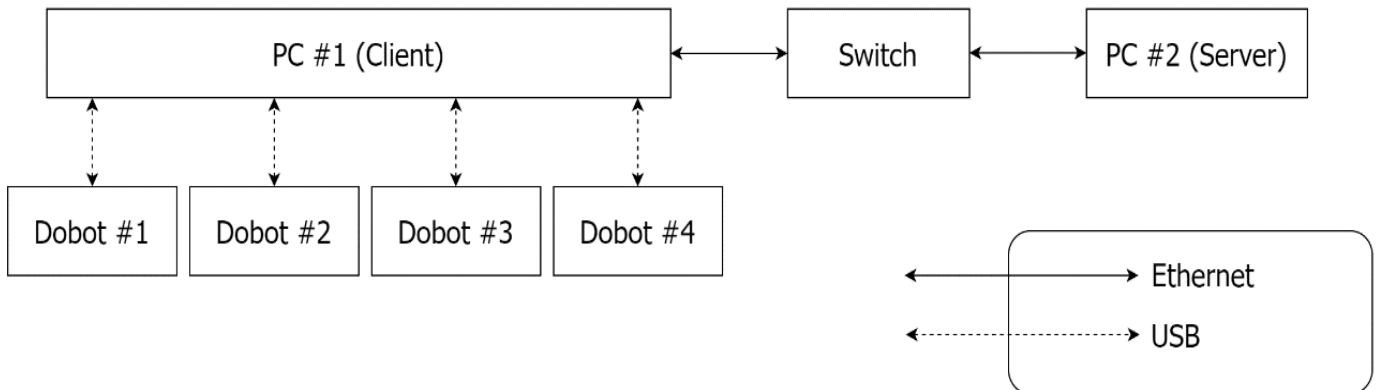


Fig. 4. Dynamic segment structure based on robotic manipulators.

B. Coordinate Generation

The robotic arm's movement limit system was taken into account to create random coordinates. This plan uses the difference between two hemispheres to calculate space, along with limits on the movement of the manipulator itself. Using the four half-planes introduced in Fig.5, any combination of these half-planes can be used to get the coordinates (X, Y, Z). The limitations of the three coordinates as introduced in Table III.

TABLE III. LIMITATIONS OF THE COORDINATES OF THE ROBOTIC ARM'S MOVEMENT AREA

Coordinate	Limits
X	$X \in (-120, 320)$
Y	$Y \in (-300, 320)$
Z	$Z > 0$

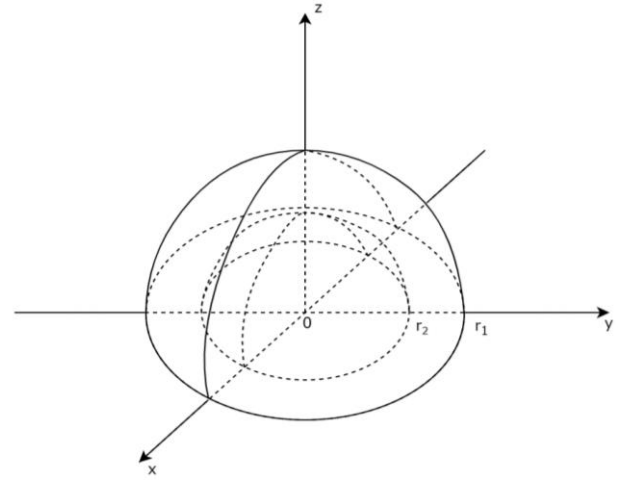


Fig. 5. Dobot magician movement restrictions.

The generated coordinates should meet the requirements introduced in Eq. (1) to be included in the valid list.

$$r_1^2 < x^2 + y^2 + z^2 < r_2^2, \quad r_1 = 300, r_2 = 180 \quad (1)$$

Where r_1 and r_2 are the radiuses of the outer and inner spheres, respectively, calculated using the manipulators' known coordinate plane (X, Y, Z). The outer sphere, with the radius r_1 , is denoted as Ω_1 , while the inner sphere, with the radius r_2 , is denoted as Ω_2 .

Additionally, coordinates must be generated to simulate the deviance of one of the four robots from the general swarm trajectory at time Δt . In order to achieve this, we rotate the

manipulator artificially to a different half-plane by setting the corresponding coordinates directly. Then, after the intended movement to the last of them, produce coordinates for movements only in the new half-plane after the time Δt . We used Dobot Studio to find the turn's coordinates. This fulfills the abovementioned parameters while allowing the robotic arm to travel freely to a new half-plane.

In addition to the previously mentioned requirements, the maximum and minimum values of X, Y, and Z were determined using factory software to construct coordinates. Shifting the manipulator to a different plane changes the X, Y, and Z limits. We receive 2 CSV files with moving coordinates as the program's output. The server then processes these files. Figure 6 displays fragments of the received files with the coordinates.

CoordinatesRandom	CoordinatesRandomForRejection
223,119,-25,15,0	223,119,-25,15,0
222,110,75,15,0	222,110,75,15,0
232,-137,49,15,0	232,-137,49,15,0
221,-25,148,15,0	221,-25,148,15,0
250,-72,130,15,0	250,-72,130,15,0
233,37,117,15,0	233,37,117,15,0
201,133,37,15,0	201,133,37,15,0
232,-136,48,15,0	232,-136,48,15,0
219,-13,64,15,0	197,-112,79,15,0
207,-56,-24,15,0	143,-196,74,15,0
228,38,109,15,0	58,-239,67,15,0
207,-119,93,15,0	13,-219,113,15,0
224,23,92,15,0	61,-253,74,15,0

Fig. 6. Fragments of the received coordinates.

The manipulator movement characteristics must be considered when solving the given problem. The robot travels in a straight line between coordinates, occasionally resulting in an overshoot. To avoid this issue, we produce two coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) for each loop iteration and add a requirement that verifies the plane going through the two points acquired does not encroach upon the sphere Ω_2 . This circumstance enables the robotic arm to stay inside the permitted positions in space.

C. Clint-Server Model

Two modules with the program code can be found in the server component. The first module examines the coordinates file for the first three robots (CoordinatesRandom.csv), and the second module examines the coordinates file for the fourth robot that deviates from the norm (CoordinatesRandomForRejection.csv). Only the names of the files used for analysis and the ports used to link the server with the client to which the robots are connected differ in these modules' code. Three sets of client socket addresses are defined in the program's main() function.

A socket is created using the socket.socket (socket.AF_INET, socket.SOCK_DGRAM) command, which is used to send datagrams using the UDP protocol (the first three robots use the official UDP ports 8000, 8880, and 8888, while the fourth deviant robot uses the official UDP port 8400). The code to read the coordinates from the file is run after creating the socket, and the data array it returns is provided to the function to transmit commands to client sockets. Each socket is closed when the loop is completed.

Four client programs make up the client portion of the software package. Each robot listens to a different port in order to maintain command parallelism. Moreover, each robot must be initialized on a different USB port to implement the chosen solution. Every client program produces a distinct socket that is linked to the appropriate port in order to receive orders from the server. As a result, the port for the USB connection between the robot and the client computer is chosen at the start of the program execution. The robot is connected to it for further control inside the *initialDobot()* function utilizing the common set of *DobotDllType* library functions designed for this operation. In the event of a successful connection, the robot's usual operating parameters are set. In the event of an initialization error, the module terminates, and an error message is sent to the console.

The robot startup function is initially called on the port given at the program's beginning in the main() function. A socket is then constructed and defined to the client computer's external IP address and the designated port. The socket is currently listening, which means it watches for messages (i.e., commands). The arriving data is decoded after configuring the buffer size of messages of the socket. An OK code is transmitted to the server socket in response to the successful reception of the message. The signaling flow of the proposed code is presented in Fig.7.

The robot moves when the received coordinates are relayed to it. After sending all the coordinates to the robot, the client PC should obtain its current position, then determine how it compares to the one the server needs. The *ERROR* code is provided to the server if the locations do not match; otherwise, the *OK* code is performed. The socket is closed when the loop is finished. Fig.7 shows a description of how the program's job is done. Only the port numbers on which the socket is created and the port numbers on which the robot is defined vary between client programs.

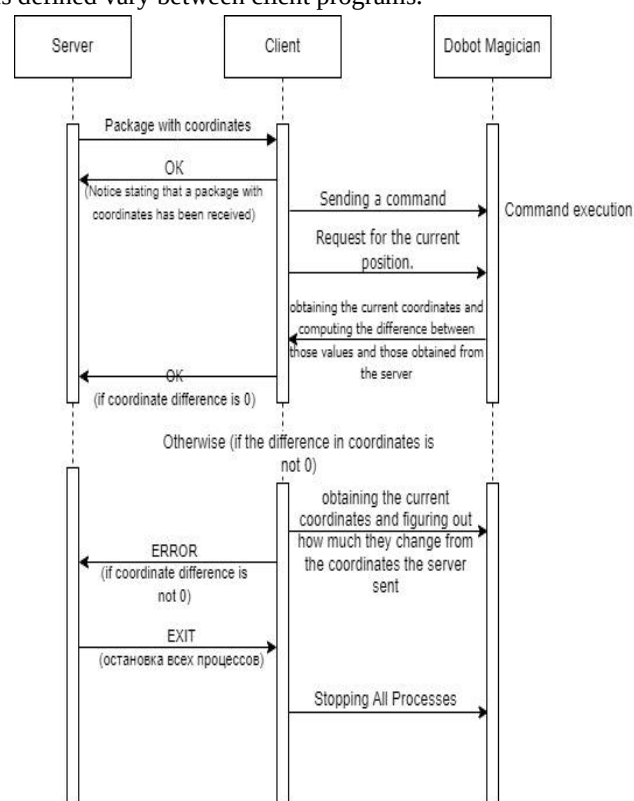


Fig. 7. Signaling flow of the proposed logical code.

D. Logical part of the testbed

The prototype's logical component was developed using Raspberry Pi 4 Model B microcomputers, which are little machines with an ARM Cortex-A72 processor clocked at up to 1.5 GHz and 8 GB of RAM. The Linux operating system is compatible with this microcomputer, ensuring optimal performance [22]. The Raspberry Pi 4 Model B is an effective and adaptable platform for a wide range of applications that can be used in both domestic and commercial settings. In the large-scale nebula model, the Raspberry Pi is a microprocessor on which one or more microservices with particular capabilities are installed. Microservice migration techniques can also be implemented using Raspberry.

The core network uses a deployed orchestrator to manage and monitor network parts. Kubernetes (K8s) is used to automatically launch, scale, and manage containerized applications in a cluster [23]. Applications can be executed in isolated environments with the help of containers. The management of containers and their deployment on a cluster of devices may be automated with Kubernetes, which streamlines the development process. Kubernetes is the ideal choice for our booth because it also offers scalability and fault tolerance. K8s offers a general API that enables you to run applications on a cluster using different container platforms like Docker. By polling only the master node and gathering network statistics, the offered API will enable the production and monitoring of each cluster node.

Docker and all the necessary Kubernetes software (kubelet, kubeadm, and kubectl) were installed on each Raspberry Pi. The child nodes (hence referred to as workers) and the main node (master-node) are both monitored by the Kubelet service, which begins and operates on each cluster node. Kubeadm is a tool used to manage a Kubernetes cluster as well as to configure nodes. To transmit commands to a Kubernetes cluster, you require the Kubectl program. The cluster worker nodes were set up and connected to the master node after the appropriate components had been installed. Our microcomputers operate as worker nodes, and our server serves as a master node. Fig. 8 presents the cluster's organizational structure.

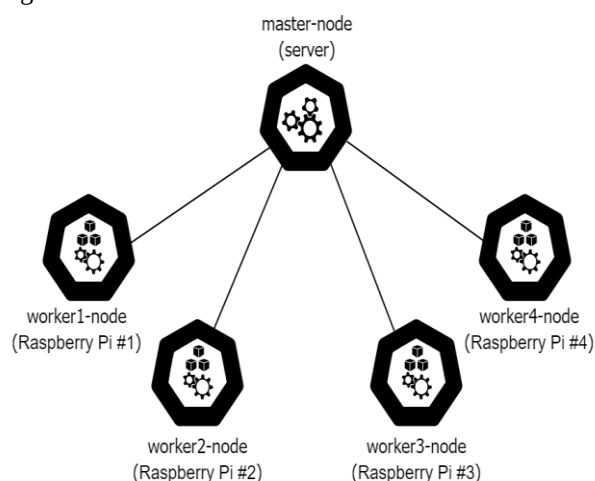


Fig. 8. Clustering scheme.

IV. MICROSERVICE ARCHITECTURE & MIGRATION SCHEME

The microservice architecture enables splitting up a monolithic application's power into sections that are part of the major service's functionality. The application developers

specify the tasks carried out by a specific microservice scheme. The result of such a system is a unique product that does not differ in functionality from the source but is more reliable and fault-tolerant in operation, with a significant increase in the performance and scalability of the entire system. The group of microservices into which the application is divided functions as a single unit, where each element interacts with all other elements. Fig. 9 presents a comparison between traditional architecture and microservice architecture.

Each microservice can be implemented using kernel-level virtualization (i.e., containerization). Organizing the architecture of various environments in one processing unit without needing a hypervisor is called containerization, a specific type of virtualization. In the conventional virtualization method, separate virtual environments are built within the hypervisor infrastructure. This method works primarily because the hypervisor's primary duty is ensuring that guest operating systems communicate with the host operating system.

Environments are created using the kernel's capabilities while using containerization. This procedure separates the kernel's capacity into parts, each only accessible by a single container, preventing the use of the system's other resources [24]. By excluding the hypervisor's impact on the system, which manifests as a decline in the performance of newly formed services and the trouble of packaging and delivering applications, a new stage in the development of virtualization has made it feasible.

In the proposed scheme, microservices are created as containers, each holding a specific portion of a major service's functionality. Planning for instances where individual services fail, even those beyond their control, is vital to provide better system reliability. An example of such cases is when the logical component of the system is in motion during the work, and one robot unit deviates from the group of others.

In this case, the tasks should be migrated to maintain the system and prevent failure. Thus, we developed a microservices migration mechanism in order to prevent this scenario. The migration scheme is introduced to maintain the service while moving between different coverage areas and to improve system performance and resource usage.

Migrating microservices offers many benefits that can be summarized as follows [25, 26].

- **Load balancing:** Migration assists in distributing the network load among all the computing power if it is severely out of balance. The service switches from an overloaded host to a more powerful or less loaded one.
- **Fault tolerance:** Failures must be foreseen and proactively handled to avoid their negative effects on the performance of the system and the running applications. Task migration provides a way for fault tolerance by providing another path for data offloading.
- **Power administration:** Migration makes it possible to distribute microservices to the fewest number of active servers dynamically. In order to save energy, inactive servers (i.e., those with no offloaded tasks) can be switched to sleep mode.

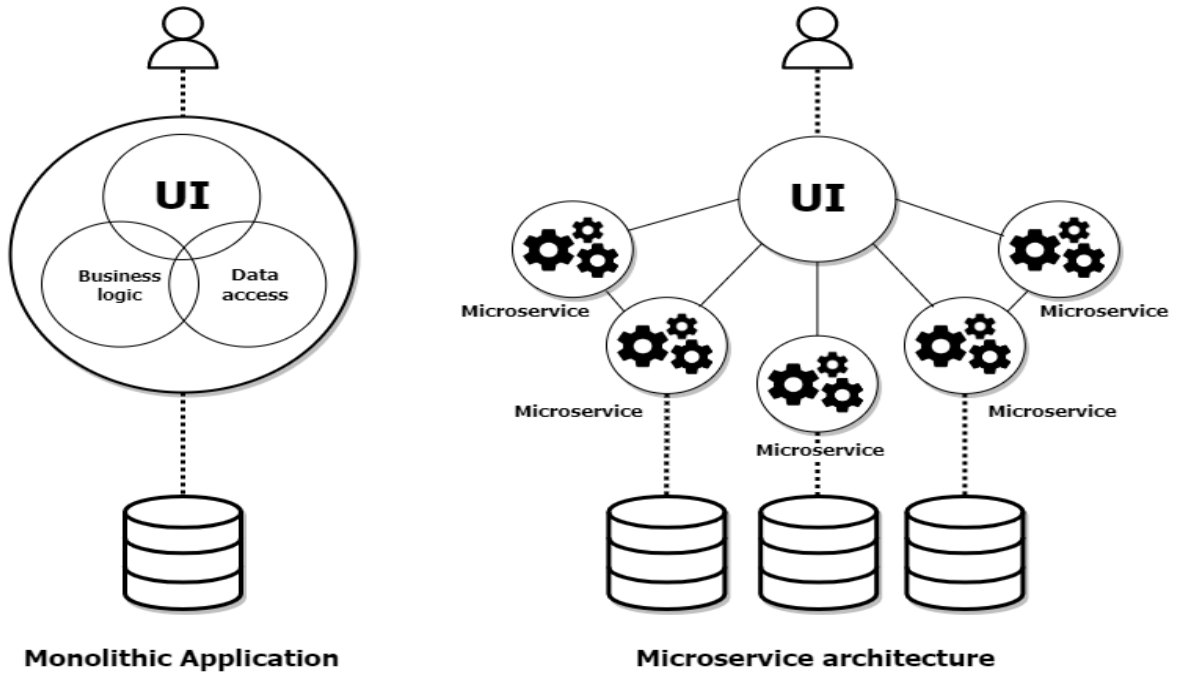


Fig. 9. Traditional and microservice application architecture.

- Prompt maintenance of the system: Suppose a particular server has to be updated or maintained. In that case, all of the microservices hosted there can be moved to a different server in the user's access zone, and the required maintenance can be carried out simultaneously on both servers.

When migrating a microservice live over a wide-area network (WAN), the online services must remain while the processor state, memory pages, and disc storage are moved from a source server to a remote destination server. Pre-copy and post-copy are the two most common migration procedures used concurrently. While the microservice (i.e., container) is still running on the source server and creating memory pages (modified memory pages), pre-copying first copies all pages of memory and disc storage to the destination server. The freshly created pages must be iteratively copied to the destination at a pace quicker than the rate at which changes are made in memory after copying the contents of the disc storage. Finally, the remaining pages and the processor state are transferred to the destination server, the source container is resumed on the new server, and the container is halted on the original server.

On the other hand, post-copy first suspends the container on the source server, moves its processor state to the destination server, resumes the container on the new server, and then migrates memory pages and disc storage. Memory pages and disc storage are actively transferred to the destination server only once since the container does not cause dirty pages to be generated on the source server.

Pre-copying, by its capacity to recover the state of the container at the source, even if the migration fails, clearly triumphs in terms of reliability. Due to the source server's lack of memory page generation, which necessitates an iterative update memory transfer, the post-copy has a quick migration time.

The post-copy technique is used in our model since it must migrate fast without the user recognizing when the moving part leaves the coverage region.

The selection of the destination server is challenging with the considered migration scheme. Selecting the optimum server for migration is an optimization problem that can be solved using practical swarm optimization (PSO) [27].

V. OPTIMIZING MIGRATION SCHEME

A. Problem Formulation

The optimization problem seeks the optimum selection of hardware required for the migration processes (i.e., optimum powerful server). The problem is a minimization problem that can be defined as follows.

$$\min_S f(x) \quad (2)$$

Where $f(x)$ is the objective function, and S is the optimum server for the migration process. The proposed objective function of the optimization problem is introduced to achieve the fastest microservice response to a request, maximum reliability and availability, and load balancing. Consequently, the objective function incorporates the server's current CPU load (in%) and RAM usage (in%). It can be defined as follows.

$$f(x) = \sum_{i=1}^n \mu_i \cdot \phi_i \quad \forall i \in \mathbb{R} \quad (3)$$

Where $f(x)$ is the objective function (calculated in %), μ_i is the weight of the i^{th} parameter, ϕ_i is the i^{th} parameter that describes the server's state, and n is the number of selected parameters. We consider only two parameters for the proposed system: CPU usage and storage requirements. Both parameters are given equal attention (i.e., equally weighted); thus, the objective function is modified as follows.

$$f(x) = 0.5 \cdot \text{CPU} + 0.5 \cdot \text{RAM} \quad (4)$$

B. Particle Swarm Optimization

The PSO is a metaheuristic algorithm that simulates how a group of birds behaves when seeking the best position in

space to approach a food source. The algorithm addresses optimization issues, such as determining a function's lowest or maximum value.

An initial set of "particles" with individual positions and speeds in the multidimensional space is created at the beginning of the procedure. Each particle moves across space, adjusting its speed and position iteratively to arrive at the food position (i.e., optimum solution). The particles update their location in space by exchanging information with other particles in the swarm, using Eq.(2) [27].

$$v_{i,j} = v_{i,j} + c \cdot r_1 \cdot (p_{i,j} - x_{i,j}) + c \cdot r_2 \cdot (p_{g,j} - x_{i,j}) \quad \forall r_1, r_2 \in [0,1] \quad (5)$$

$$x_{i,j} = x_{i,j} + v_{i,j} \quad (6)$$

Where $v_{i,j}$ is the speed of the particle i , $p_{i,j}$ is the best position found, $p_{g,j}$ is the best-found position over all particles, $x_{i,j}$ is the position of the particle i in space, c is a constant with value 2, and r_1, r_2 are random numbers.

The value of the objective function is calculated for each available device using equation (4) to identify the best device for migration. The next step is to select a device from the available devices with the minimum objective function.

VI. CONCLUSION

Along with the advancement of information technology, the demand for society to incorporate novel concepts in this field into everyday life is growing. Unfortunately, the current infrastructure established by conventional communication networks is inadequate for introducing new technologies. This procedure requires gradually modernizing a well-established, global infrastructure that has expanded worldwide for years. Consequently, research on microservice architecture networks and fog computing is growing in significance. Using these and other related solutions, you can create more efficient algorithms and models to satisfy the increasing demands of contemporary technologies. Using the under-consideration architecture to research uRLLC services would accelerate bridging the gap between current data and the Tactile Internet concept's specifications.

ACKNOWLEDGMENT

The studies at St. Petersburg State University of Telecommunications. Prof. M.A. Bonch-Bruevich was supported by the Ministry of Science and High Education of the Russian Federation by grant number 075-15-2022-1137.

REFERENCES

- [1] C. X. Wang et al., "On the road to 6G: Visions, requirements, key technologies and testbeds," *IEEE Commun. Surv. Tutor.*, pp. 1–1, 2023.
- [2] M. M. Azari et al., "Evolution of non-terrestrial networks from 5G to 6G: A survey," *IEEE Commun. Surv. Tutor.*, vol. 24, no. 4, pp. 2633–2672, Fourthquarter 2022.
- [3] C. Yeh, G. D. Jo, Y.-J. Ko, and H. K. Chung, "Perspectives on 6G wireless communications," *ICT Express*, vol. 9, no. 1, pp. 82–91, 2023.
- [4] L.-H. Shen, K.-T. Feng, and L. Hanzo, "Five facets of 6G: Research challenges and opportunities," *ACM Comput. Surv.*, 2022.
- [5] D. Calandra, F. G. Praticò, A. Cannavò, C. Casetti, and F. Lamberti, "Digital twin- and extended reality-based telepresence for collaborative robot programming in the 6G perspective," *Digit. Commun. Netw.*, 2022.
- [6] A. A. Ateya, A. Muthanna, A. Koucheryavy, Y. Maleh, and A. A. El-Latif, "Energy efficient offloading scheme for MEC-based augmented reality system," *Cluster Comput.*, vol. 26, no. 1, pp. 789–806, 2023.
- [7] D. M. Hilty et al., "A review of telepresence, virtual reality, and augmented reality applied to clinical care," *J. Technol. Behav. Sci.*, vol. 5, no. 2, pp. 178–205, 2020.
- [8] F. Hernandez, M. Waechter, and A. C. Bullinger, "A first approach for implementing a telepresence robot in an industrial environment," in *Advances in Human Factors and System Interactions*, Cham: Springer International Publishing, 2021, pp. 141–146.
- [9] M. Osama, A. A. Ateya, S. Ahmed Elsaid, and A. Muthanna, "Ultra-reliable low-latency communications: Unmanned aerial vehicles assisted systems," *Information (Basel)*, vol. 13, no. 9, p. 430, 2022.
- [10] G. P. Sharma et al., "Towards deterministic communications in 6G networks: State of the art, open challenges and the way forward," *arXiv [cs.NI]*, 2023.
- [11] V. Benavente, L. Yantas, I. Moscol, C. Rodriguez, R. Inquilla, and Y. Pomachagua, "Comparative analysis of microservices and monolithic architecture," in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2022, pp. 177–184.
- [12] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice Architecture Reconstruction and Visualization Techniques: A Review," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2022, pp. 39–48.
- [13] A. Razzaq and S. A. K. Ghayyur, "A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges," *Comput. Appl. Eng. Educ.*, 2022.
- [14] L. Tawalbeh, F. Muheidat, M. Tawalbeh, M. Quwaider, and A. A. Abd El-Latif, "Edge enabled IoT system model for secure healthcare," *Measurement (Lond.)*, vol. 191, no. 110792, p. 110792, 2022.
- [15] F. Al-Doghman, N. Moustafa, I. Khalil, N. Sohrabi, Z. Tari, and A. Y. Zomaya, "AI-enabled secure microservices in edge computing: Opportunities and challenges," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1485–1504, 2023.
- [16] B. Costa, J. Bachiega Jr, L. R. de Carvalho, and A. P. F. Araujo, "Orchestration in fog computing: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 2, pp. 1–34, 2023.
- [17] A. Volkov, K. Proshutinskiy, A. B. M. Adam, A. A. Ateya, A. Muthanna, and A. Koucheryavy, "SDN load prediction algorithm based on artificial intelligence," in *Communications in Computer and Information Science*, Cham: Springer International Publishing, 2019, pp. 27–40.
- [18] K. Kaur, V. Mangat, and K. Kumar, "A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture," *Comput. Netw.*, vol. 217, no. 109281, p. 109281, 2022.
- [19] W. Lv et al., "Microservice deployment in edge computing based on deep Q learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 1–1, 2022.
- [20] A. Bandi, "A review towards AI empowered 6G communication requirements, applications, and technologies in mobile edge computing," in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, 2022, pp. 12–17.
- [21] "DOBOT Magician robotic arm," Automate. [Online]. Available: <https://www.automate.org/products/shenzhen-yuejiang-technology-co-ltd-dobot/dobot-magician-robotic-arm>. [Accessed: 07-May-2023].
- [22] Raspberry Pi Ltd, "Raspberry Pi 4 Model B specifications –," Raspberry Pi. [Online]. Available: <https://www.raspberrypi.com/products/raspberrypi-4-model-b/specifications/>. [Accessed: 07-May-2023].
- [23] "Production-Grade Container Orchestration," Kubernetes. [Online]. Available: <https://kubernetes.io/>. [Accessed: 07-May-2023].
- [24] S. Muhizi, A. A. Ateya, A. Muthanna, R. Kirichek, and A. Koucheryavy, "A novel slice-oriented network model," in *Developments in Language Theory*, Cham: Springer International Publishing, 2018, pp. 421–431.
- [25] R. Pérez, M. Rivera, Y. Salgueiro, C. R. Baier, and P. Wheeler, "Moving microgrid hierarchical control to an SDN-based Kubernetes cluster: A framework for reliable and flexible energy distribution," *Sensors (Basel)*, vol. 23, no. 7, p. 3395, 2023.

- [26] M. Adeppady, P. Giacccone, H. Karl, and C. F. Chiasserini, "Reducing microservices interference and deployment time in resource-constrained cloud systems," *IEEE Trans. Netw. Serv. Manag.*, pp. 1–1, 2023.
- [27] M. Jain, V. Saihpal, N. Singh, and S. B. Singh, "An overview of variants and advancements of PSO algorithm," *Appl. Sci. (Basel)*, vol. 12, no. 17, p. 8392, 2022.