Journal of Theoretical and Applied Information Technology
31st July 2025. Vol.103. No.14
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

LATENCY-AWARE NETWORK SLICING USING DISTRIBUTED RESOURCE ALLOCATION AND REGIONAL ORCHESTRATION

SAIF SAAD ALNUAIMI
Department of Cybersecurity Engineering, College of Information Engineering, Al-Nahrain University,
Jadriya, Baghdad, Iraq
E-mail: saif.alnuaimi@nahrainuniv.edu.iq

ABSTRACT

Network slicing has emerged as a fundamental enabler for delivering diverse services with heterogeneous requirements in next-generation communication systems. However, most existing approaches focus on either bandwidth or computational resource allocation in isolation, often relying on centralized architectures that struggle with latency, scalability, and information privacy across distributed network components. This paper addresses this gap by proposing a novel distributed network slicing architecture that jointly optimizes bandwidth and compute resources. The core innovation is the introduction of a regional orchestrator (RO) a new control plane entity positioned between base stations (BSs) and cloud nodes to coordinate localized resource allocation while preserving system privacy and scalability. We develop a distributed resource allocation algorithm based on the splitting model (dra-SM) to efficiently manage joint resource distribution without centralized control. Simulation results show that our approach significantly reduces overall network latency by approximately 15%—compared to single-resource slicing, while also achieving faster convergence and service-specific latency guarantees. This work contributes a scalable, low-latency solution for real-world deployment of joint network slicing across decentralized infrastructures.

Keywords: *Data Slicing, Cloud, Network Slicing, Bandwidth Consideration, Resource Allocation.*

1. INTRODUCTION

The general consensus is that 5G will involve much more than just an improvement in throughput and capacity, which are purely physical performance measurements. A significant shift away from the usual data-oriented design will be made in favor of an architecture that is service-oriented and more flexible [1]. A single set of physical network resources may now be used to provide a broad variety of services, each with its own specific set of requirements thanks to the Service-Based Architecture (SBA) that the 3GPP has introduced [2]. The main idea is to integrate software-defined networking (SDN) with network functions virtualization (NFV) to virtualize network components into network functions, each of which is built of a functional building block that utilizes a separate network resource [3]. A network slice is a collection of network function sets may then be used to instantiate each type of service. The core of SBA has been viewed as network slicing, which can adapt to a variety of service requirements and application situations [4].

Cloud computing has lately been pushed as one of the major elements of 5G by both industry and standards organizations to enable new, computationally intensive applications open up fresh commercial opportunities and increase income. Cloud computing is an approach of offloading computationally intensive work that is located than UEs are positioned closer to massively-scale cloud data centers, which are often situated in faraway locations. Computerized clouds are made up of a lot of tiny computer servers, also known as cloud nodes [5]. Cloud computing service providers like Amazon and Microsoft can set up cloud computing networks. Mobile network operators (MNOs) can also deploy it into their network infrastructure. Network slicing, which makes use of both communication and computing resources, has recently generated a lot of interest [6]. Allowing each slice to be supported by both resources may improve the overall UE experience. Additionally, this can open the way for resource balancing across multiple network parts and for the creation of future services with demanding computational and latency specifications [7].

Motivated by the limitations of existing approaches, this study proposes a novel distributed framework designed to enable scalable and latency-aware network slicing through joint coordination of bandwidth and computational resources. Unlike traditional solutions that rely on centralized control and complete system visibility, our architecture introduces a new control plane entity the Regional Orchestrator (RO) to facilitate localized decision-making without the need to exchange sensitive information between base stations and cloud nodes. The study develops a distributed optimization algorithm based on the splitting model (dra-SM), allowing for efficient coordination across the system while preserving scalability and privacy. Simulation results demonstrate that the proposed method achieves approximately 15% improvement in overall network latency compared to models that manage only one type of resource. Additionally, the algorithm converges more rapidly than several conventional optimization methods, validating its effectiveness for real-world, latency-sensitive network scenarios. This work therefore contributes to the field by addressing critical challenges in distributed resource slicing, offering a practical and robust solution applicable to future communication systems such as 5G and 6G.

The capacity to distribute resources across several network slices with different resources is great but it also comes with many additional difficulties. First, several service providers often handle various resources. As a result, it is typically difficult for them to exchange or share confidential information, such as the accessibility of resources and traffic patterns. Second, since the network infrastructure for cloud computing and communication may be dispersed over Wide scale, unacceptably long coordination lag periods and high communication overhead may occur from centralized coordination and administration [8]. Last but not least, each UE has the capacity to simultaneously request a range of services with a range of features offered by a range of resources. It's still unclear how to create the best algorithm for rapidly and reliably allocating diverse resource combinations to support many network slices [9].

This study analyzed distributed network slicing for network systems that consists of comprises an operational cloud computing network and a network of base stations (BSs) delivering wireless communication services and handling computationally demanding tasks. For the support of many network slices, we take into consideration the combined resource allocation of BSs' bandwidth and cloud nodes' computing power [10]. Our main goal is to lessen the total latency that ends UEs experience, which takes into account both queuing delay at cloud nodes and communication delay in wireless networks linking UEs and BSs. An algorithm for distributing resources has been put forth. The paper demonstrates how the suggested method may ensure adequate performance for every type of service it supports while reducing the network's overall average latency. The significance of this paper's contributions is outlined in the following:

1) To allow distributed network slicing in a large network, a new distributed framework based on regional orchestrator (RO) technology is being developed.

2) The use of a distributed optimization algorithm based on the distributed resource allocation approach is based on the splitting model (dra-SM) has been proposed in order to distribute coordinate the resource allocation of both bandwidth of BSs and computational resources of nodes without the need for the exchange of any sensitive information between BSs and cloud nodes. We demonstrate that the suggested method can achieve global optimality at an $O(1/t)$ pace.

3) Various real-world scenarios have been offered for simulation and in-depth performance analysis. According to our findings, joint slicing, which makes use of both bandwidth and computing resources, versus network slicing with a single resource, lowers overall latency by around 15%.

Existing network slicing approaches often rely on centralized architectures, single-resource allocation, and full information sharing making them unsuitable for distributed, large-scale, and privacy-sensitive environments. This study fills that gap by introducing a distributed architecture with regional orchestrators and a dra-SM-based algorithm to jointly allocate bandwidth and compute resources, enabling low-latency, scalable, and privacy-preserving network slicing.

2. RELATED WORKS

Data storage, processing, and other services are supported by the cloud computing paradigm, which offers a wide range of services to its clients. In fact, cloud providers supply all the resources required, allowing for pay-per-use internet access to the services supplied. In view of these facts, it would be essential to deploy the suggested services automatically and optimally can reduce expenses, raise QoS, or even boost service reuse. In reality, while creating composite services, a carefully done mapping of previously optimizing the new offers requires the use of current services in a significant way [11].

The positioning of application components and data is referred to as the SaaS positioning Problem (SPP) in this context associated with them in a group of computing and storage servers, as an example of a service deployment issue. Numerous placement options are created by the various correlations of the resources that the servers have to give. Therefore, the best placement strategy is one that guarantees a given level of acceptable QoS and other criteria [12].

The optimization of resource utilization is, in fact, the next issue to be considered after choosing from among the available servers. Because the placement method has a direct impact on that, it qualifies as a strategic combinatorial issue [13]. To maintain SaaS performance and needs fulfilment, restarting the SPP on the pre-selected servers involves installing software and data components. SaaS may be provided in a composite format in order to satisfy these criteria. Both processing and storage servers are used to deliver three composite SaaSas. It should be noted that the third composite SaaS can only be made up of application components, which is how the deployed SaaS may be constructed.

The main objective of the SaaS has been to decrease the total execution time (TET) majority of SPP work it has been seen when surveying the SPP works. Some of these initiatives have just focused on this goal, while others have taken into account additional goals like cost reduction and load balancing. Additionally, the limitations considered or even the optimization aims might be used to categorize previous works. In addition to these traits, it is possible to categorize earlier SPP techniques by application components or data chunks) that make up a SaaS [14].

The findings and goals of the researchers serve as the primary motivators for categorizing SaaS placement techniques. Actually, the majority of them have discussed the SPP from two perspectives. While some studies have interpreted software as a collection of application and data components analogous to the well-known component placement problem (CPP), as a group of software components, has been the perspective of others. The optimization goals are taken into consideration in our categorisation. While previous researchers have worked with provider-related constraints the researchers' main goal was to enhance the placement scheme by focusing on the customer-related limitations (such as cost and total execution time). These constraints included resource usage, load balancing, and inter-task communication [15][16].

Several prior studies have addressed aspects of resource allocation and service provisioning in cloud-enabled networks. Early works have focused on centralized cloud architectures and static resource provisioning, primarily targeting either computational offloading or bandwidth management. For example, SaaS positioning problems (SPP) have been tackled by optimizing application placement across distributed servers [11–14], emphasizing performance metrics like total execution time (TET), load balancing, and cost minimization. However, these studies typically neglect real-time service dynamics and the integration of both communication and compute resources.

In the domain of network slicing, many approaches rely on centralized SDN controllers [6, 8], which limit scalability and introduce high coordination delays—especially in geographically distributed infrastructures. Furthermore, while SDN and NFV technologies are central to 5G evolution, existing solutions often assume full knowledge sharing, an unrealistic assumption in multi-operator or privacy-sensitive environments. Other studies using heuristic or machine learning-based models for cloud resource allocation [2, 4, 5] provide improvements in energy efficiency and provisioning policies, but they are generally limited to single-layer resource management and lack support for joint optimization across layers communication plus compute. Moreover, security and privacy concerns in resource sharing are often overlooked.

The gap is further reinforced by the lack of distributed frameworks capable of achieving low-latency, joint resource slicing, and decentralized optimization under limited information conditions. These limitations form the foundation of the problem addressed in this study.

This critique builds a logical path from the current literature to your work's problem statement, which is then addressed through your distributed RO architecture and dra-SM model.

3. METHODOLOGY

3.1 System model

Fig 1 shows a network system with a collection of S = 1 nodes, a collection of S ={1…S} BSs, and a collection of F = {1…F} cloud nodes. Each BS in a cellular network within its designated service area offers services. Consider that each UE may only request a total of N distinct services. Assume that N = {1…N} represents the entirety of all supported services. We assume that BSs may communicate a specified amount of data for each type of service, referred to as a task unit, which cloud nodes may then handle. For instance, a service that processes video or audio can make use of several data units to process and send each video or audio clip. Give each task unit of service type n a data size of d_n. A fixed bandwidth designated as s for all s∈S has been assigned to each BS, and each cloud node is capable of handling a maximum of µ_f task units per second for every f∈F.Assume that k_sn ~ P(λ_sn), where sn is the expected number of task units received and λ_sn is the Poisson distribution of the nth service task unit arrival rate at BS s.

We take into account cooperative resource allocation for various network slices in this article. Using a technique called network slicing; Virtual Network Functions (VNFs) are created by virtualizing physical resources. Each VNF may be further broken into smaller components and placed in a shared software container in order to make the network functionality easily released and reused by numerous service instances. Slice units are the smallest building block that may be utilized in VNFs for network slices. There are several slice units that can make up a network slice. Slice units are separated from one another. Thus, the launch and dynamic scaling of each network slice may be done without disrupting other running services.

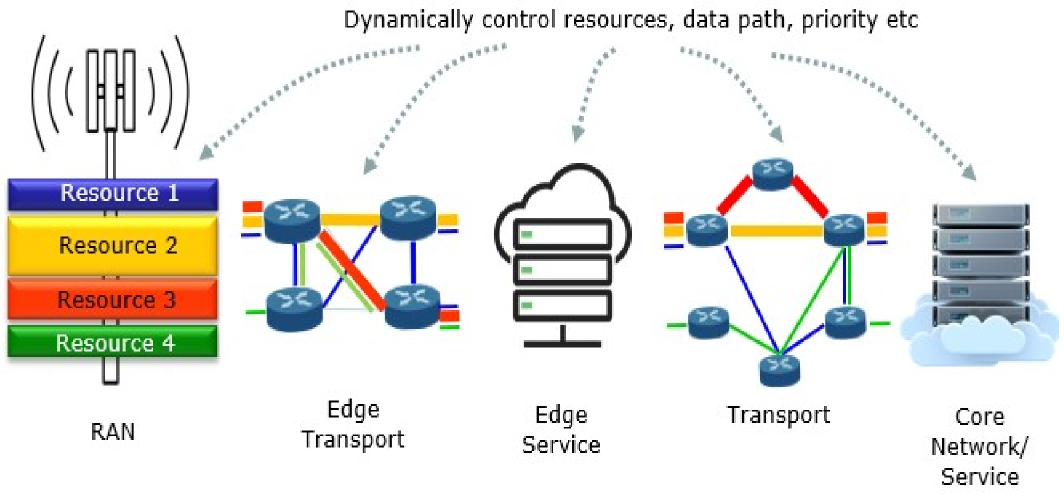

Figure 1: Network slicing

4. System Model

The goal of this study is to decrease service response times for each kind of service, which may result in queue delays at cloud nodes as well as communication delays for task unit transfer from user equipment to cloud nodes. Let's start by thinking about the communication latency. It should be noted that in many real-world networks, UEs and BSs can link through wired or optical fibre, which often gives a far greater data rate than wireless connectivity. As a result, the paper use a standard setup and don't take into account the communication lag between BSs and cloud nodes. According to a commonly used setup, the communication latency for sending each unit of the work with a certain bandwidth 0≤ b_sn<β_s provided by BS s for service type n may be written as:

$$p_{sn} = \frac{d_n}{b_{sn}.\log\left(1 + h_{sn}\frac{w_{sn}}{\sigma_{sn}}\right)} \qquad (1)$$

Where w_{sn} is the required transmission power to transmit the task units for service type n from the UEs to BS s and h_{sn} is the channel gain between the BS and the corresponding UE for service type n. The term "σ_{sn}" refers to the volume of noise that is received at BS s. The pace at which cloud nodes can process data and at which tasks arrive can have an impact on the queueing latency at the cloud node.

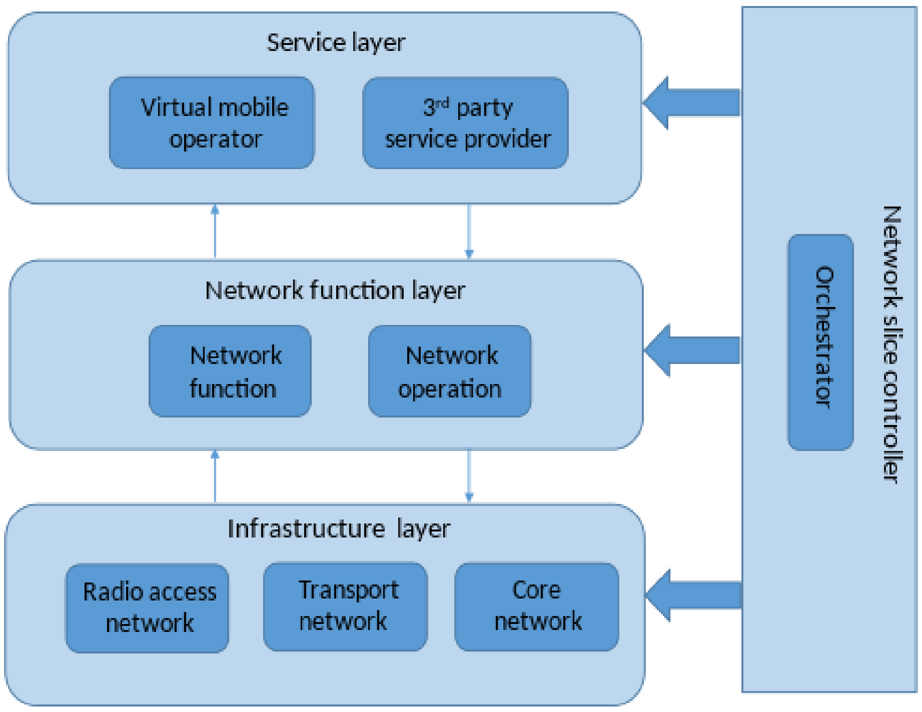

Figure 2: Distributed network slicing

Assume that μ_{sn} is the maximum amount of processing power that cloud nodes can devote to handling the nth kind of service that BS s provides to the connected UEs. We presumptively characterize the task units controlled by cloud nodes as $M/M/1$ queuing under a regularly employed configuration. We may depict the n^{th} kind of service's queuing latency in the BSs' coverage area as:

$$q_{sn} = \frac{1}{\mu_{sn} - \lambda_{sn}} \quad (2)$$

When (1) and (2) are combined, the total service response time for the n^{th} service type provided by BS s may be shown as:

$$t_{sn} = p_{sn} + q_{sn} \quad (3)$$

3.3.1. Network slicing

As was already indicated, the following two issues must be addressed in order to swiftly and reliably allocate resources across networks of computing and communication:

1) Since physical resources can be widely dispersed across a wide area, unacceptably high latency and pointless communication overhead may be produced by a centralized resource management and control system.

2) Different service providers may own the infrastructure for communication networks and cloud computing networks. As a result, neither of these systems can exchange proprietary information.

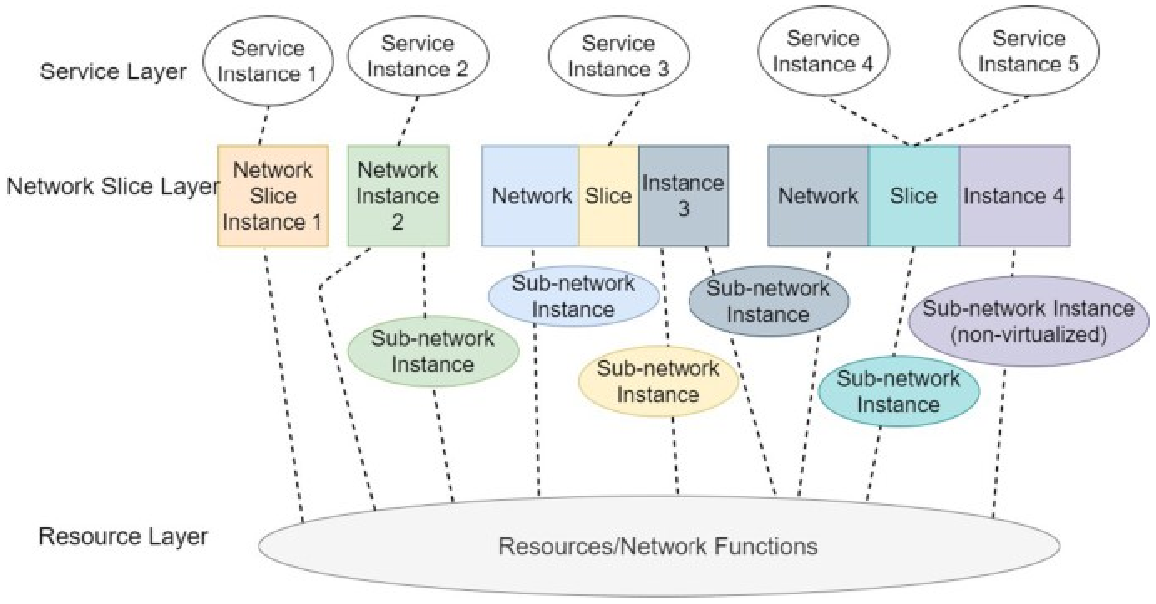

Figure 3: Service layers for network slicing

Adding other centralized SDN control plane frameworks to a distributed environment would not solve the aforementioned problems. In reality, several published studies have noted that SDN controllers have been created with a primary emphasis on establishing and sustaining connections across virtual mesh networks as well as controlling the routing of data traffics. Even in a mobile setting, it may be utilized to keep the network connected and keep the services running. To control the processing resources of cloud computing networks, however, it cannot be used. Additionally, OpenFlow is dependent on a central SDN controller to manage network resources and can only offer static paths to each SDN switch.

This paper proposed a distributed network slicing architecture based on a novel control plane entity, RO, positioned between the communication network and cloud computing network, to enable the fine-grained control of resources across both network systems. The whole coverage area has been divided into a number of smaller sub-regions under this arrangement. A few close-by BSs and cloud nodes make up each of these sub-regions, and they may be quickly joined to one another through local wire line connections. In order to administer a set of VNFs made up of neighboring computing and communication resource units, RO may be installed in each sub-region. Only local VNFs inside each sub-region can be controlled and instantiated by a single RO for network slicing. Every time a UE asks a service task from a BS, the resource requests will be submitted by the BS to the RO. The RO will then communicate with the BS making the service request, neighbouring cloud nodes, and other ROs to build the necessary network slices. Furthermore, the RO be in charge of BSs and cloud node path reservations and service traffic routing. If there are uneven traffic loads in two or more adjacent sub-regions, two or more ROs can cooperate and collaboratively alter their local VNF production and distribution rates. Fig 3 shows the illustration of our suggested architecture.

The network slicing architecture developed by the 3GPP mandates that certain resources be set aside and allocated for various types of services that are provided to guarantee that resources are always accessible whenever a service request is made. The RO utilized in the 3GPP framework is examined. Service type n task units at BSs, each RO must provide the number of computing resources (λ_{sn}) at cloud nodes and the amount of bandwidth (b_0) that will be available. The paper has the following restrictions because the requested service instances will be supported using the resources that have been set aside throughout this period:

$$b_{sn} > b_0 \quad (4)$$

$$\mu_{sn} > \lambda_{sn} \quad (5)$$

This study's main focus is on resource allocation and network slicing over a predefined time period, where a group of local BSs' maximum bandwidth and a set quantity of local cloud nodes' processing power have been allotted for a group of supported types of services. The network slicing and dynamic resource allocation will be kept for later research. We take into account the following restrictions:

1) Bandwidth limitation: Limit the overall bandwidth provided to each BS to β_s. To put it another way, BS s is not permitted to allocate more bandwidth overall than β_s to all pending service jobs. In general, the RO must set aside enough resources even without being aware of the exact number of task units that will be deployed in the future. However, the empirical probability distribution of the task arrival rate enables the RO to forecast the possible number of task units. Therefore, for the great majority of potential activities, the RO can dependably reserve adequate resources to support the performance-guaranteed services. The probability that fewer Type n service task units than a specified threshold arrived at BSs amount is how we establish the confidence level θ_{sn} denoted by the symbol $\theta = Pr(k_{sn} \leq \theta_{sn})$. For instance, if $\theta - 0.9$, the RO wishes to hold back enough resources to confidently satisfy the requests of every UE. We can see that θ the task arrival rate k_{sn} Cumulative Distribution Function (CDF) is identical to. This allows us to write $\theta_{sn} = CDF_k^{-1}(\theta, \lambda_{sn})$, where $(\bullet)^{-1}$ denotes the inverse function. Then we may limit the amount of bandwidth that BSs can provide a set N of all services that are supported kinds as follows.

$$\sum_{n \in N} \theta_{sn} . b_{sn} \leq \beta_s \quad (6)$$

The total amount of computational power γ that may be distributed to all cloud nodes in a subdivision is limited. The cumulative sum of allowed computer resources for the services cannot be higher than. The next computational resource limitation is as follows.

$$\sum_{s \in S} \sum_{n \in N} \mu_{sn} \leq \gamma \quad (7)$$

A maximum tolerated latency, denoted as T_n, is also assumed for each supported kind of service so that we have $t_{sn} \leq T_n$, $\forall s \in S, n \in N$. Open-ended and adaptable architecture is what we advise. You might use it for network slicing that makes use of several resources spread out across a large geographic area. In this article, network slicing design is considered to maximize the RO. In order to dynamically distribute computing and bandwidth resources, the BSs and cloud nodes can work with the RO. In order to improve the following issue, we focus on creating a distributed technique.

$$\min_{\{b_{sn}\}\{\mu_{sn}\}} \sum_{s \in S} \sum_{n \in N} t_{sn} \quad (8)$$

$$b_{sn} > b_0, for\ all\ s \in S, n \in N \quad (9)$$

$$\mu_{sn} > \lambda_{sn}, for\ all\ s \in S, n \in N \quad (10)$$

$$t_{sn} > \bar{T}_n, for\ all\ s \in S, n \in N \quad (11)$$

3.3.2. Optimized network slicing

As stated earlier, we must carefully choose the resources made available to each network slice if we want to reduce the total latency encountered by end UEs. The problem requires, even though this may result in unacceptable high latency and communication overhead, that the appropriate amount of bandwidth and processing power for each type of service be jointly determined using global data such as the anticipated number of task units that will arrive and the computational capacity of each cloud node. We must build a distributed optimization algorithm with the following design objectives in order to address the aforementioned problems as well as the combined network slicing challenge:

1) Distributed Optimization with Coordination: The proposed optimization strategy must be able to segment the overall issue into several smaller problems that can each be resolved by a BS using its local knowledge. In order to get at the global optimum resource allocation solution, the resolution of numerous sub-problems can then be coordinated through the RO.

2) Privacy protection: It's feasible that BSs and cloud nodes won't want to exchange sensitive information, such as bandwidths, with one another anticipated arrival rates for task units, and processing capacity.

3) Fast Convergence: Over time, changes might be made to the BSs and cloud nodes connected to each RO. Consequently, the algorithm must quickly arrive at the overall optimal solution.

We provide distributed resource allocation approach based on the splitting model ($dra - SM$)-based distributed optimization technique. $dra - SM$is more suited to addressing issues with inequality-constrained optimization in a decentralized way as compared to conventional convex optimization methods. Furthermore, it is feasible to safeguard the sensitive data of BSs and cloud nodes thanks to the decomposition-coordination process of $dra - SM$. Sadly, issues with two blocks of variables are the limit for problems that can be handled by standard $dra - SM$ procedures. In order to overcome problem, with the aforementioned goals, we offer a distributed $dra - SM$ method with partial variable splitting known as distributed resource allocation. This approach splits the Lagrangian dual issue into S sub-problems, where each might be handled by a separate BS using its regional competence. The BSs will deliver their preliminary findings to the RO, who will compile them and offer coordinated input.

To integrate the restrictions with the objective function, let's first take the same approach and introduce a collection of $S + 1$ indicator functions. Specifically, for constraints that can be separated across different BSs, we define $G_s = \{b_s, \mu_s : b_{sn} > b_0, \mu_{sn} > \lambda_{sn}, t_{sn} \leq T_n, P_{n\in N}\theta_{sn} \bullet b_{sn} \leq \beta_s, \forall s \in S, n \in N\}$ as the feasible set corresponding to BS s where $b_s = < b_{sn} >_{n\in N}$ is the vector of bandwidth allocated by BS s for each type of services and $\mu_s = < \mu_{sn} >_{n\in N}$ is the vector of processing power allocated for each type of services connected to BS s. If $x_s = < b_s, \mu_s >, and\ for\ all\ s = S,$ then the S indicator functions are as follows:

$$I_{g_s}(x_s) = \begin{cases} 0 & x_s = g_s \\ +\infty & x_s \neq g_s \end{cases}, for\ all\ s \in \quad (12)$$

We may also create an indicator function $IG(\mu)$ for constraint that cannot be separated.

$$I_g(\mu) = \begin{cases} 0 & \mu \in G \\ +\infty & \mu \neq G' \end{cases} \quad (13)$$

Where, G is the half-space denoted by the Eq.$G = \{\mu: P_{s\in S}P_{n\in N}\mu_{sn} \leq \gamma\}$, $\mu - [\mu_1, \mu_2, ..., \mu_S]$. By incorporating the previously mentioned indicator functions, IGs, and IG, the original joint network slicing issue with a set of inequality requirements may be changed into the following form without inequality restrictions.

$$\min_{\{x_s, z_s\}} \sum_{s\in S} \{f(x_s) + I_{G_s}(x_s)\} + I_G(z) \quad (14)$$

Where, the newly added auxiliary variable $z - [z_1, z_2, ..., z_S]$and $f(x_s) = P\ n \in Ntsn.$ The augmented Lagrangian is provided by:

$$L_P = (x, z, \wedge) = \sum_{s\in S} \{f(x_s) + I_{G_s}(x_s)\} \quad (15)$$

Where, the enhanced Lagrangian parameter is ρ and the dual variable is Λ. So, we can demonstrate the resulting fact.

Theorem 1: The given augmented Lagrangian is convex and somewhat separable among x_s.

Proof: Convex sets include the feasible set for issue and it's containing set G_s, as well as set $G_s, \forall s \in S$, half space G, and their intersection. Convexity is established instantly by this. In addition, we demonstrate that $f(x_s)$ second derivative is always positive inside the feasible set of problems indicating that the function is convex. We can next demonstrate that $L_\rho(x, z, \Lambda)$ is convex by showing that summing maintains convexity. The augmented Lagrangian is rewritten in the following way to demonstrate that it is partly separable:

$$L_P = (x_1, \dots, x_S, \Lambda) = \sum_{s \in S} \{f(x_s) + I_{G_s}(\quad (16)$$

We can see from Eq. (12) that for $s \in S, L_\rho(x_1, \dots, x_S, z, \Lambda)$ may be partly separated over x_s. The proof comes to an end here. The following is how we may transform issue into a two-block form.

$$x^{k+1} = arg \min_x \sum_{s \in S} \{f(x_s) + I_{G_s}(x_s)\} + \quad (17)$$

$$z^{k+1} = \arg\min I_G(z) + \frac{\rho}{2} \left| |x - z^k + \Lambda \right. \quad (18)$$

$$\Lambda^{k+1} = \Lambda^k + x^{k+1} - z^{k+1} \quad (19)$$

Where, k is total iterations. We may break down Eq. (13) into several smaller issues using the partly separability of $L(x, z, \Lambda)$, which each BS may address using its local expertise. The following sub-problem is resolved by each BS in particular.

$$x_s^{k+1} - f(x_s) + I_{G_s}(x_s) + \frac{\rho}{2} \left| |x - z^k - \right. \quad (20)$$

The point $x^{k+1} + \Lambda^k$ is projected onto the half space G which is equal to saying:

$$z^{k+1} = \prod_G (x^{k+1} + \Lambda^k) \quad (21)$$

Here, $\prod_G(.)$ stands for the projection onto halfspace G. In Approach 1, a thorough description of our suggested approach is provided. The combined network slicing issue has a global optimum solution and the suggested algorithm converges to it at a rate of $O(1/t)$.

Proof: We suggested $dra - SM$ algorithm's convergence feature closely resembles that of the conventional technique because the centralized x —update and the distributed subproblems presented are equivalent. Due to space restrictions, we have omitted the specifics from this section.

Algorithm 1: $dra - SM$

```
Initialize: Every BS with initial variable $x_s^0$ and RO selects variables $k = 1$
Choose maximal iterations $K > 0$;
While $k \le K$ do
    Every BS performs simultaneously do:
    Update $x_s^{k+1}$ and perform RO;
    Allocate bandwidth for all arrived task based on $b_s^{k+1}$;
    After $x_s^{k+1}$ are attained and RO do;
    Revise variable $z^{k+1}$;
    Update successive variable $\Lambda^{k+1}$;
    If stopping criteria fulfils
        Break;
    End if
    Transfer sub-vectors $\Lambda_s^{k+1}$ and $z_s^{k+1}$ to BS;
    $k = k + 1$;
end while
```

3.3.3. Genetic Algorithm

Since the 1980s, a family of evolutionary hill-climbing algorithms known as genetic algorithms (GAs) have been widely utilized on a range of search and optimization problems in the fields of engineering and operations research. It has been demonstrated that they are effective in solving several challenging issues in these situations, large state spaces, a lack of state knowledge, and non-stationary environment. GA is inextricably coupled to a certain encoder, which converts each viable tactic into a unique binary sequence (code) of a specific length. A haphazard collection of codebook sequences, or what is known as the initial population, is made at the initialization stage. Each potential approach is assessed to determine its fitness, or the importance of the desired target function. New populations are then repeatedly created based on the preceding population's fitness ratings. Every iteration in a typical GA comprises of the following three phases in order:

Reproduction: Depending on how well it performs at this stage, each unique approach from the previous population gets replicated into a new set. The repetition rate of the duplicated set's copies varies in direct proportion to the origin's fitness value in the preceding population. It is possible for the better candidates to expand through reproduction while the worst underperformers are eliminated since the size of the reproduced set is the same as that of the previous population.

Crossover: All sequences in the replicated set are matched at random in this stage. A random subsequence swapping opportunity exists between each pair. This results in the creation of novel sequences at random, a situation where each "child" has the opportunity to mix and pick up advanced "genes" from its two "parents." Faster convergence to the optimum and higher likelihood of local convergence result from a higher possibility of swap (crossover rate).

Mutation: Through the process of mutation, each candidate sequence has the potential to have one or more random components reversed which promotes exploring the codebook. A higher incidence of mutation or a higher number of mutated bits reduces the likelihood of local convergence while simultaneously making the convergence meander more, increasing the likelihood of wandering from the world's best.

GAs provides model-free advantages and may be used online, just as the Q-Learning method. However, unlike the majority of reinforcement learning systems, GAs, like Q-Learning, put less emphasis on the reward value of each individual action and more emphasis on various overall strategy "fitness" values that have been quantified. Due to the fact that in some applications it might be challenging to choose the fitness function effectively, this is occasionally seen as a disadvantage of GAs. Another typical criticism of GAs is that it may be difficult to build the strategy encoder, particularly in situations where continuous-valued decisions must be taken. None of these flaws, however, have any relevance in the context of the present study because: 1) Fitness functions are available for end business KPIs such is the average network utility over the long period indicated. The NO only makes binary decisions in a finite state space, resulting in the inherent binary encoding of every strategy.

3.3.4. GA Slicing Techniques

Only if the network operators (NO) active slice set at the moment falls inside the area of freedom, can the NO make a free binary decision as was previously explained. In every other scenario, the NO must reject each incoming request for the formation of a new slice. We calculated the limited enumerable set known as the NO's area of discretion D as the initial stage in encoding slicing algorithms. To represent each slicing strategy by a unique $||D||$-bit-long binary sequence and enable the enumeration of all possible slicing strategies in this codebook, we assigned the integer range $[0, ||D|| - 1]\ to\ D$. This mapping is displayed. Using the aforementioned code, we developed a cutting-edge genetic algorithm-based slicing approach optimizer. The following describes how this optimizer works in an online manner.

a) Initialization

A starting population P^1 of possible tactics is present in the pre-generated codebook and is randomly picked by the NO to be kept in the background for "virtual" operation. In terms of actual functioning, the NO employs a randomly generated beginning strategy, d^1.

b) Fitness Evaluation

At the start of each evolution term, the NO records its active slice set and creates an evolution term $T > 1$ that is greater than one (normalized to one operation period). As the network is running, the NO responds to each incoming tenant request in accordance with the slicing strategy it is presently using, while also making a "virtual" choice for each person in the background in accordance with every potential plan in the available population. The NO keeps track of the simulated utility for each and every suggested approach during each operations period. Every strategy's fitness in the current population P_j is assessed at the conclusion of the j^{th} evolution term as its produced or simulated average utility across the T operations periods:

$$\bar{u}_i^j = \frac{1}{T} u_{\Sigma, p_i^j(t)}, for\ all\ 1 \leq i \leq P, j \qquad (22)$$

c) Evolution

The candidate in P^j with the highest fitness is initially chosen to update the strategy for use in the following evolution term:

$$p^{j+1} = \arg\max_{p_i^j \in P^j} \bar{u}_i{}^j \qquad (23)$$

The normalized fitness values, a reproduction $\tilde{p}^j$ of P_j are then produced. $p_i^j \in P^j$in$\tilde{p}^j$ is an arbitrary copy number.

$$A_i^j = round\left\{P * \frac{\tilde{u}_i^j + \epsilon}{\sum_{i=1}^{P} \tilde{u}_i^j + P_\epsilon}\right\} \qquad (24)$$

Where, ϵ is a low value to reduce inaccuracy in the unlikely event where $\sum_{i=1}^{P} \tilde{u}_i^j = 0$. The components in $\tilde{p}^j$ are then shuffled and coupled, with a chance for the crossover operation to be performed on each pair. After the crossover, each potential tactic in the new population is adopted through a certain number of rounds β of mutation, during which time there is a distinct possibility γ, that candidate's lone random bit strategy will be inverted. Using the resulting set of strategies, for virtual functioning in the following phase of progression, the population P^{j+1} is updated.

1. Numerical results and discussion

We built up a network system in Dublin with 285 BSs and 285 cloud nodes that can process three different types of services—text, audio, and video—are, to assess the efficacy of our suggested network slicing design. We simulate many scenarios to show the performance of our proposed distributed network slicing architecture three sites, ranging from urban cores to suburban areas, as shown in Fig 3. It is assumed that each BS has a 30MHz bandwidth set aside for it and that nodes in the cloud can only handle 180 task units per second. We assume that the processing and bandwidth resources reserved by cloud nodes and BSs in the same area are equivalent. Slices of computing resources and bandwidth are two different network slicing situations that we compare respectively—each of which only makes use of one kind of resource with the combined slicing incorporating both the capacity of cloud nodes and the bandwidth of BSs.

First, we assess how well Algorithm 1 performs in terms of convergence. A comparison between the interior-point approach and the first algorithm we propose ($dra - SM$) for various iteration counts is shown. Applications of the interior-point method in communication network systems are many. We can see that within the first few rounds, our suggested the first algorithm can approach the lowest latency with high precision. Compared to the interior-point approach, it can provide substantially quicker convergence performance. In all three of these types of places, we can see that the joint slicing architecture we've suggested outperforms existing network slicing architectures, demonstrating the broad geographical applicability of our design. Since the features of the three designs are the same across all areas, we only go into depth about the area 1. When each BS has a variable amount of bandwidth available, we fix the processing power allotted to each cloud node in Fig 6 together with the value in order to assess the service response time.

We can see that when the complete bandwidth is allocated, the service response time gets faster. Furthermore, we can see that bandwidth slicing works better when the bandwidth of BSs is limited, computational resource slicing is preferable to that. The slicing of computing resources' service response time, however, tends to reduce considerably more quickly than that of the bandwidth slicing when the bandwidth of each BS grows. This is so because we fixed the computing resource in our simulation. When each BS has a restricted bandwidth, as in this situation, communication delay predominates over total latency. Because of this, adopting bandwidth slicing to reduce communication latency may be more effective than boosting cloud nodes' computational power to reduce the time it takes for all services to react. Overall latency will be dominated by the queuing delay after each BS has enough capacity. The slicing of computing resources will be more helpful in this situation to speed up the service response time.

Each cloud node's assigned processing power was compared in three network slicing scenarios for the analysis of service response times, in Fig 6, We establish a fixed amount of bandwidth for each BS and value. We can see that when processing power is set aside for each cloud node, the service response time gets faster. Similar to this, we see that bandwidth slicing performs worse than computational resource slicing when the computing resource of the cloud nodes is constrained. However, once each cloud node's computational capacity is reached, bandwidth slicing begins to slow down more quickly than the slicing of computing capacity. This is so that Fig 7 bandwidth will always be fixed. When a cloud node's processing capacity is constrained, the overall latency is dominated by the holding period. As each cloud node's processing power rises, communication latency starts to overtake other delays in terms of the overall length of time.

For the purpose of examining the service response time at various values of θ, we set the processing power and bandwidth allotted for each cloud node and BS. As we can see, the service response time gets θ faster as the business expands. this is because as it rises θ, there are more task units overall that BSs must convey. Due to the longer communication lag caused by this, the service response time will be longer. Furthermore, we see that θ the bandwidth slicing service response time increases much more gradually than the computational resource slicing the lengthening of service response times. This is due to we modify the processing capacity and bandwidth allowed for each BS and cloud node, respectively increases the number of task units for each service from θ each BS in this case. When θ little, each work unit is has access to enough bandwidth, and the queuing delay predominates in terms of total latency. The bandwidth allotted to each work unit is constrained as increases θ, and communication delay will begin to predominate the total latency.

4.1. Analysis with GA

An NO with a one-dimensional normalized resource pool was taken into consideration in order to make a concise and persuasive presentation with minimal computing complexity:

$$r - [r_1] - 1 \tag{25}$$

It accepts $M = 1$ *and* $N = 2$ as two separate slices. As a result, the resource cost vector is also one-dimensional and set to $c_1 = c_2 = [0.3]$ for each slice type cn. Then, as shown in Tab 1, it is feasible to obtain a tiny decision-making space D with a size of 12 and a tiny resource-planning space S with a size of 10. According to this specification, there are $2^{12} = 4096$ possible slicing techniques in total. We assumed that the two slice types' periodical utilities were equal to two and one, respectively, indicating that slice type 1 has a utility-efficient twice as high as slice type 2. We also established $T = 6$ utility generation operations periods as the length of an evolution term. We hypothesized that requests for slice production came in a Poisson manner where k_n is the number of requests that arrive over a certain operations time for each slice type $n \in [1, 2]$:

$$P(k_n)\ request\ arrived = e^{-\lambda}\frac{\lambda_n^{k_n}}{k_n!}, fo \tag{26}$$

We believed that each slice of type in the interim was $n \in \{1, 2\}$ had an exponentially distributed random lifespan (normalized to one operations period):

$$f(\tau_n = t_n) = \frac{1}{\mu_n} e^{-\frac{t_n}{\mu_n}}, \qquad for\ all\ t_n \tag{27}$$

Three possibilities for services with different parameter settings $[\lambda_1, \lambda_2, \mu_1, \mu_2]$ were developed for our simulations, as shown in Tab 1.

Table 1 Resource feasibility

S **Element**	***D*** **Element**
[0,0], [0,1], [0,2], [0,3], [1,0] [1,1], [1,2], [2,0], [2,1], [3,0]	[0,0,1], [0,0,2], [0,1,1], [0,1,2] [0,2,1], [0,2,2], [1,0,1], [1,0,2] [1,1,1], [1,1,2], [2,0,1], [2,0,2]

Table 2 Model parameters

Scenario	λ_1	λ_2	λ_3	λ_4
1	0.6	2.1	2	11
2	0.4	1	3	4
3	1.1	0	3	6

4.2. Efficiency Analysis

We created two genetic slicing strategy optimizers in simulation, one with a population size of 50 and the other with 10 viable strategies in each generation to show the efficacy of our suggested method. Starting with a resource pool that was completely idle and a random population of potential tactics, each optimizer completed 20 generations of evolution. To achieve a speedy convergence, both genetic optimizers were programmed with a complete a round of mutations with a rate of mutations of $\alpha = 1$ and a crossover rate of $\gamma = 0.1$ and a rate of $\beta = 1$. For the Monte-Carlo test, we ran this simulation 500 times while monitoring the average network utility over a long period as specified in Eq. (27). By running the identical Monte-Carlo test 500 times in complete search, the global optimum out of all 4096 viable methods was found as a benchmark. Additionally, we evaluated three "naive" reference techniques to serve as benchmarks for performance comparison:

• **Greedy:** accepting every request that comes in, provided that the resource pool can accommodate it.

• **Conservative:** It makes logical to accept all requests for type 2 slices but refuse all requests for type 1 slices.

• **Opportunistic:** accepting all type 1 slice requests but refusing all type 2 slice requests for two pieces. The results are shown in Fig 3. It is evident that both genetic optimizers began with low utility levels at first, but shortly (within 4 generations) converged to effective cutting techniques with competitive results.

The genetic optimizers converged to a local maximum in the simulated progress, falling short of the global utility efficiency maximum. . Even so, both genetic optimizers beat the three static naïve reference techniques starting with the after considering 30 or 150 out of the 4096 options, or after the fourth generation of evolution, with long-term average network utilities above 90% in relation to the global optimum. Additionally, it can be seen that the two optimizers' convergence is boosted by an increase in population size by contrasting them with one another.

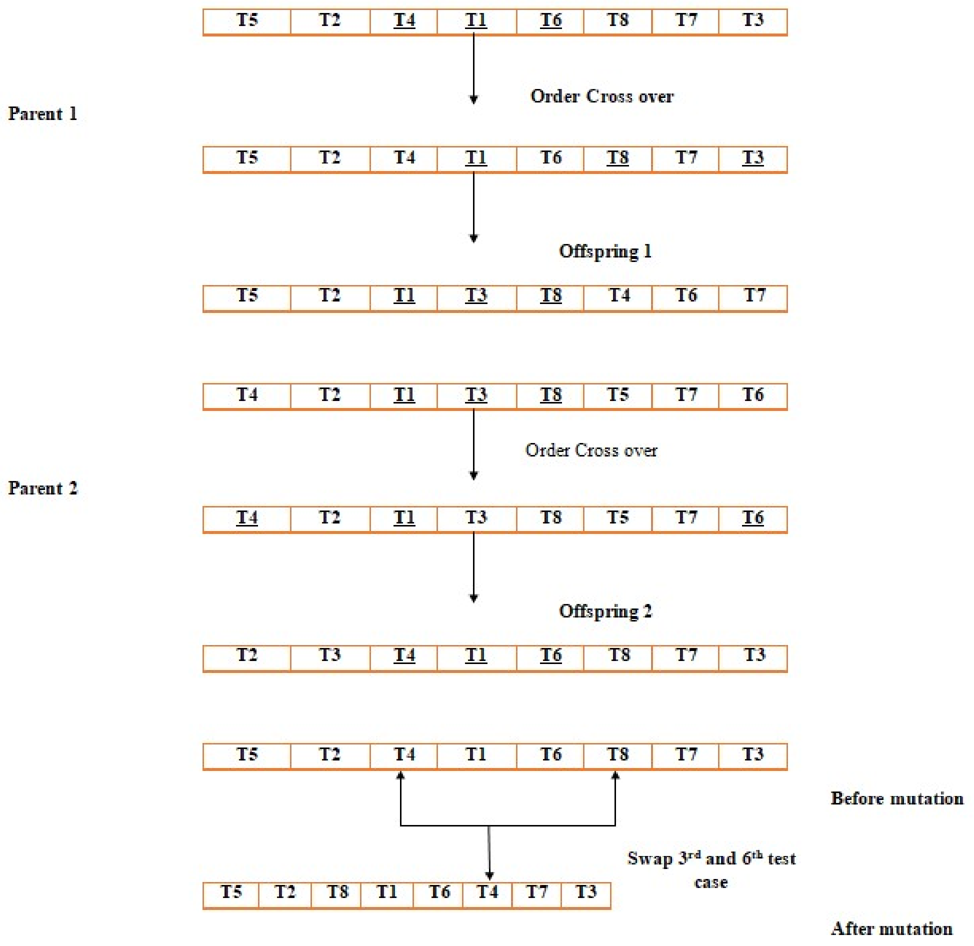

Figure 4: GA process

4.3. Population

The fact that GA allows for population evolution in each iteration, in addition to the best candidate, is a key component. The performance distribution of the 50 methods used by the genetic optimizer across numerous generations is depicted in Fig 4. It is possible to see considerable progression toward a set of general "good" strategies. This phenomenon highlights the ability of our genetic optimizer, when used in conjunction with other machine learning techniques, to produce training sets for both updating and initializing.

GAs is well renowned for being resistant to non-stationary conditions. When discussing the optimization of a slicing method, this has to do with how resource requests and slice terminations behave in a statistically time-varying manner. To assess how well our genetic slicing strategy optimizer performed in these conditions, we ran a simulation across 60 generations of evolution, or 360 operation periods. For the first 20 generations of the strategy set, the scenario was set to 1, creating a global optimum approach that is identical.

The scenario was changed to #2 for generations 21 to 40 and again to #3 for generations 41 to 60. A genetic optimizer was employed in this instance, using parameters $[\alpha, \beta, \gamma] - [1, 1, 0.1]$ and a population size of 50. Comparisons were made between its performance and the three naive reference strategies mentioned previously as well as the strategy used in scenario 1 that is global optimum. In Fig 4, the results and conclusions of 500 Monte-Carlo tests are shown. The genetic optimizer was able to respond swiftly to environmental changes and, as a result, maintained its high level of performance. The scenario-specified optimum, in contrast, performed poorly dynamically when the environment changed. Similarly, it became discovered that the environment had a significant impact on how well all static reference schemes performed.

Our genetic slicing technique optimizer was put to the test by being placed in a more complicated environment with slice scales that were substantially lower $(c_1 = c_2 = 0.03)$. In accordance with this standard, the NO has two spaces: a place for unrestricted choice D, 1122 square feet; and a space for resource viability S, 595 square feet. This results in a staggering total of 1122 for the number of its conceivable slicing techniques. The utility efficiency scales were also adjusted to $[u_1, u_2] - [0.2, 0.1]$ and the service scenario parameters were set to $[\lambda_1, \lambda_2, \mu_1, \mu_2] = [2.5, 10, 2, 10]$. Then, using populations of 10 and 50, we tried two genetic optimizers $[\alpha, \beta, \gamma] = [1, 1, 0.1]$ was the configuration for both optimizers. Once more, we employed the benchmark reference approaches "Greedy", "Conservative" and "Opportunistic" no world-wide ideal was considered since it would be computationally too expensive for those with poor local maximums to fully search for the optimum high. Both optimizers were able to fast converge within 10 generations, as shown in Fig 5, but only to get far worse local maxima than all reference techniques.

A random strategy's usual separation from the global optimum, the size of the strategy space, the number of local maxima, and all grow as a function of strategy space size, which is the origin of these phenomena. As a result, there is an increased chance of local convergence. Furthermore, since the GA begins with a random population, it is quite easy for it to converge to a low level. To increase the beginning population and to lessen early convergences at regional maximum, one or more reference techniques can be included in order to help performance deviate from the benchmark level, the original population therefore achieving the first goal. A greater mutation rate or a lower crossover rate may be beneficial for the second.

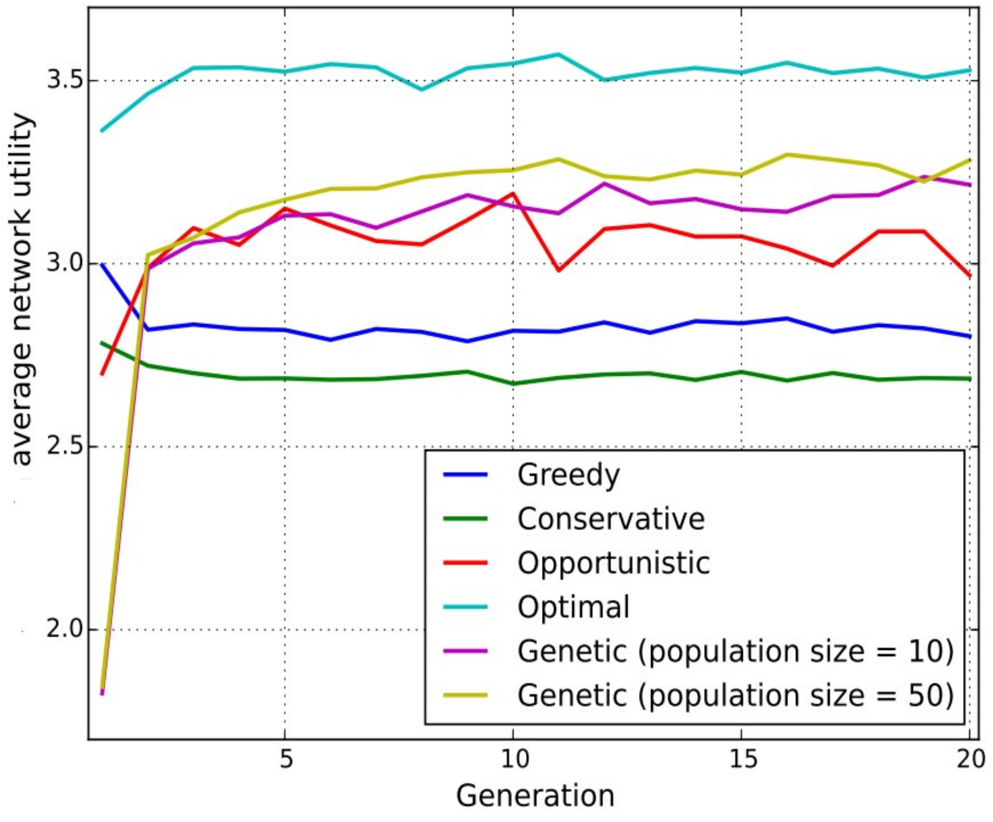

Figure 5: Generation vs. average network utility

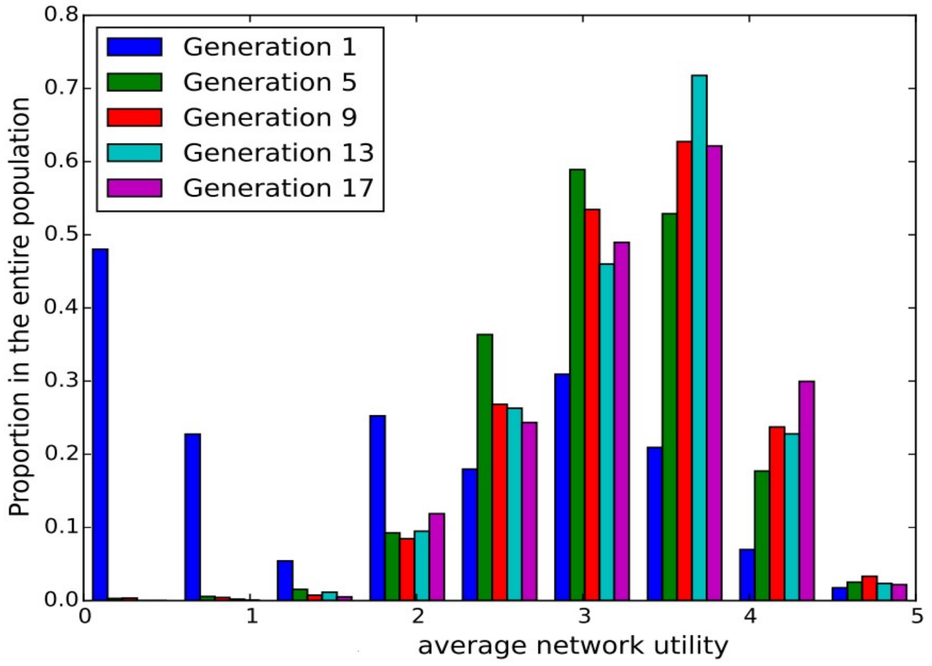

Figure 6: Network utility vs. entire population

Complexity: To prevent the random degeneration that may result from mutations, it is also a typical GA approach to protect one or more "elite" individuals from crossover and mutation operations in each generation and then immediately introduce them into the following generation. Since both optimizers' starting random populations explicitly include, the reference strategy "Greedy" we conducted the aforementioned simulation once more. Both optimizers were set up to protect the greatest member of each demographic generation and to have $[\alpha, \beta, \gamma] = [0.9, 1, 0.1]$. Fig 5 to Fig 7 presents the outcomes. It is evident that the genetic optimizers either met or, at the very least, matched the benchmark using these little improvements where both optimizers attained convergence after six generations.

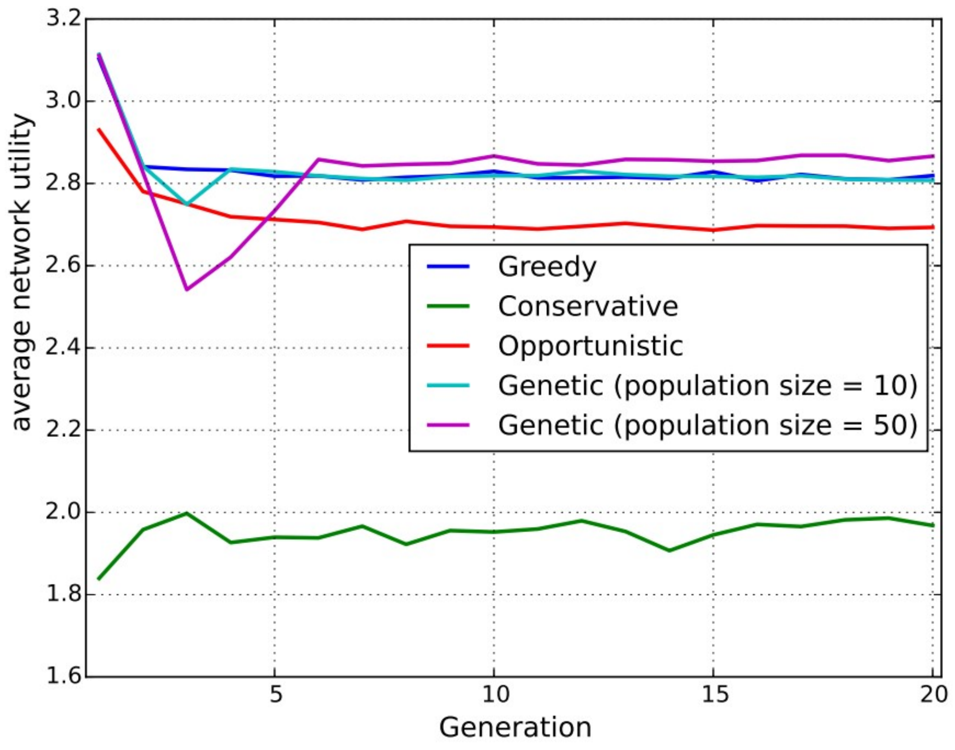

Figure 7: Generation vs. average network utility with reduced complexity

For the time being, we can state that any specific static strategy will be defeated by our genetic optimizer, despite the fact that the time needed very slightly increases with the size of the problem space for convergence. Furthermore, it's crucial to keep in mind population growth will support convergence. Since our suggested strategy is easily parallelized for the temporal complexity of our technique is unaffected by the examination of many candidate strategies within the same generation, nor is it affected by population upscaling. As a result, it is incredibly scalable and suitable for challenging practical applications.

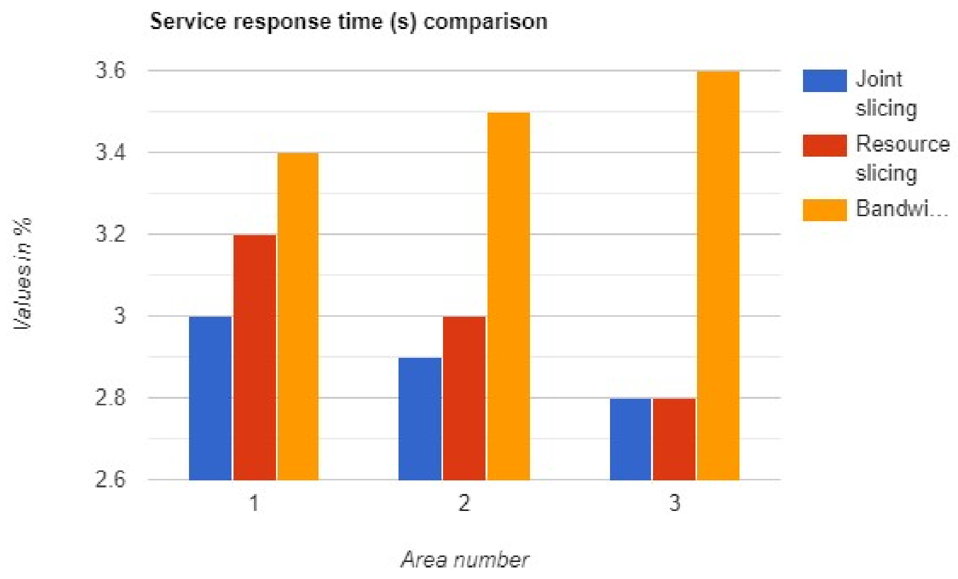

Figure 8: Service response time comparison

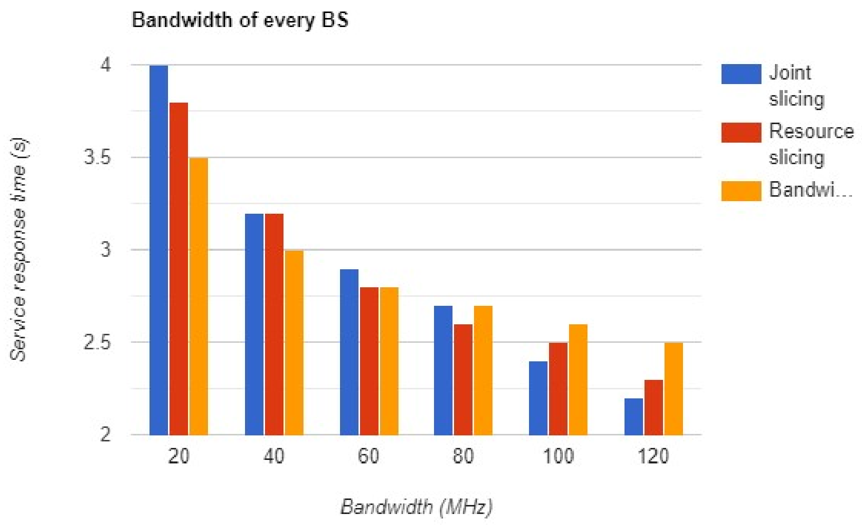

Figure 9: Bandwidth vs. Service response time (s)

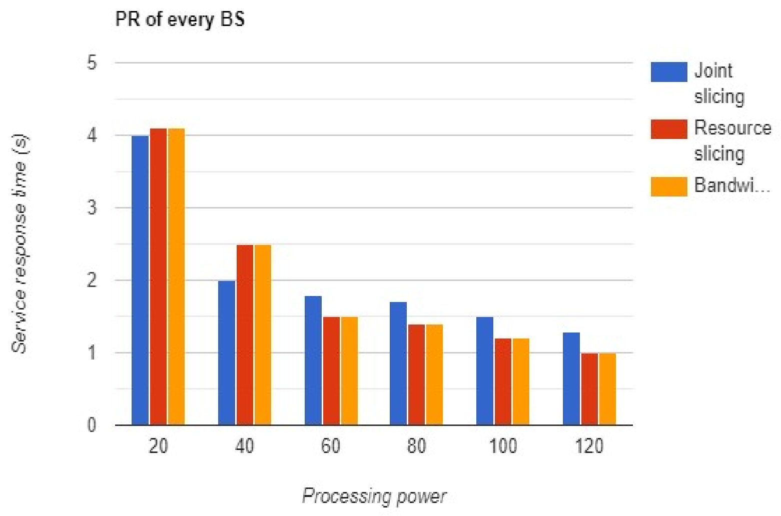

Figure 10: PR of every BS

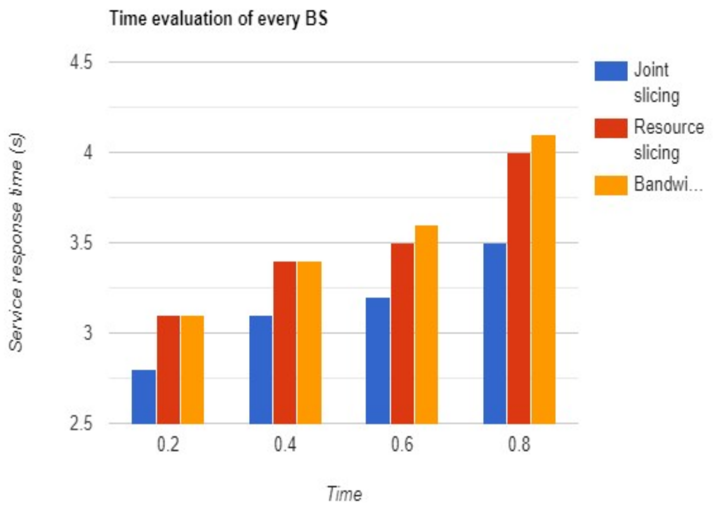

Figure 11: Time evaluation of every service response time (s)

To boost long-term network usefulness, we have introduced in this chapter a unique online genetic slicing approach optimizer. The suggested method has been put to the test using numerical simulations, and the results show a satisfying approximation to the global optimum, quick convergence, prompt adaptability to environmental changes, and high scalability (See Fig 8 to Fig 11). Furthermore, it doesn't require any prior understanding of the traffic or utility model.

This study has certain drawbacks, even if the suggested distributed architecture and dra-SM-based allocation model showed good performance in lowering latency and guaranteeing effective resource use. First, idealized assumptions like static bandwidth and consistent processing power across nodes are assumed in the simulation environment, which might not accurately represent the variability observed in real-world deployments. Second, the study ignores dynamic or real-time adjustments in favor of concentrating on a set time range for resource allocation. Third, security features like hostile node behavior or data leakage were overlooked even though privacy-preserving architecture was given top priority. To further improve the usefulness and resilience of the suggested system, further research should include adaptive, real-time optimization, more diverse node behavior, and more solid security frameworks.

5. DIFFERENCES FROM PRIOR WORK AND STUDY ACHIEVEMENTS

Unlike prior research, which predominantly focuses on centralized architectures and either communication or computational resource slicing in isolation, this study introduces a novel distributed network slicing framework that enables joint optimization of both bandwidth and compute resources. Previous approaches often assume full global knowledge and centralized control, leading to scalability issues, increased latency, and privacy concerns especially in wide-area, multi-provider environments.

In contrast, the paper proposed architecture leverages Regional Orchestrators (ROs) as intermediate control entities, enabling localized resource allocation without the need for sharing sensitive information between base stations and cloud nodes. Additionally, the use of a splitting-model-based optimization algorithm (dra-SM) allows for distributed resource coordination with provable convergence and scalability.

Through extensive simulation across diverse urban scenarios, the study demonstrates that the joint slicing method achieves approximately 15% reduction in average service latency compared to single-resource slicing. It also converges significantly faster than traditional optimization methods, making it practical for real-time and large-scale deployments. These achievements directly address the limitations identified in the literature and fulfill the need for a distributed, privacy-preserving, and latency-aware network slicing solution in future 5G and 6G networks.

6. PROBLEMS AND OPEN RESEARCH ISSUES

Despite the promising results achieved by this study, several challenges and open research issues remain that warrant further investigation. First, the current work assumes a static allocation window and does not incorporate dynamic or real-time resource adjustment mechanisms. In practical environments, network conditions and service demands are highly dynamic, necessitating the development of adaptive resource slicing strategies that can respond in real time to traffic fluctuations and user mobility.

Second, while the use of Regional Orchestrators (ROs) enables decentralized control and privacy-preserving coordination, the study does not fully explore the security implications of distributed orchestration. Future research must investigate the resilience of ROs against malicious nodes, data tampering, and denial-of-service attacks, particularly in multi-operator scenarios.

Third, the simulation environment was built on several simplifying assumptions regarding bandwidth uniformity, node capability, and service homogeneity. Real-world deployments are significantly more complex, with heterogeneous infrastructures and varying Quality of Service (QoS) requirements. Addressing these complexities will require multi-objective optimization models that can balance latency, throughput, energy consumption, and fairness.

Fourth, while this study focused on minimizing average latency, other performance metrics such as jitter, reliability, and energy efficiency remain underexplored in the context of joint slicing. Integrating these dimensions into the optimization framework will further improve its applicability to ultra-reliable low-latency communications (URLLC) and other 6G use cases.

Lastly, although the dra-SM algorithm showed faster convergence, there remains scope to enhance scalability and reduce computational overhead, especially in scenarios involving thousands of nodes. Research into lightweight distributed algorithms and edge-assisted intelligence may help scale such systems efficiently in large, complex environments.

7. CONCLUSION

For a system that combines a network of BSs that offers wireless communication services with a cloud computing network that manages computationally taxing tasks, this research goal is to investigate distributed network slicing. The core of the cutting-edge distributed system we propose is the regional orchestrator (RO), a unique control plane entity that may be positioned coordination and resource management of bandwidth and computational resources between BSs and nodes. To distribute coordinate the deployment of BS bandwidth and cloud node computing resources, additionally, a method for allocating dispersed resources has been made available. According to the outcomes of our simulations, the suggested method can rapidly approach the global optimal solution. Additionally, the performance of the network's latency may be greatly increased by using the suggested distributed network slicing design in this paper.

Beyond proposing a novel technical architecture, this study provides valuable insights for practitioners and researchers working on next-generation network systems. Readers gain an understanding of how joint coordination of bandwidth and compute resources—facilitated by a distributed architecture with regional orchestrators can substantially reduce end-to-end latency in cloud-supported communication networks. The dra-SM-based distributed optimization strategy shows that efficient resource slicing can be achieved without compromising privacy or requiring full global knowledge, which is critical in real-world, large-scale deployments. By demonstrating a ~15% latency reduction and faster convergence compared to traditional methods, this research not only addresses theoretical challenges in decentralized optimization, but also offers a practical, scalable framework applicable to 5G and future 6G networks. Ultimately, the reader walks away with a deeper comprehension of how fine-grained, regionally managed slicing techniques can enhance service delivery and QoS in highly dynamic network environments.

REFERENCES

[1] Jyoti, A., Shrimali, M.: Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing. Cluster Comput., 1–19 (2019)

[2] Lin, J., Dai, Y. Chen, X, Wu, Y.: Resource allocation of cloud application through machine learning: a case study. In: Proceedings of the International Conference on Green Informatics ICGI IEEE, pp. 263–268, August 2017

[3] Alsadie, D., Tari, Z., Alzahrani, E.J., Zomaya, A.Y.: Dynamic resource allocation for an energy-efficient VM architecture for cloud computing. In: Proceedings of the Australasian Computer Science Week Multiconference on ACSW 2018 ACM, Brisband, Queensland, Australia, pp. 1–8, January 2018

[4] Madni, S.H.H., Latiff, M.S.A., Coulibaly, Y., Abdulhamid, S.M.: September, Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. Cluster Comput. 3(20), 2489–2533 (2017)

[5] Pradhan, P., Behera, P.K., Ray, B.N.B.: Modified Round Robin Algorithm for Resource Allocation in Cloud Computing. In: Proceedings of the International Conference on Computational Modeling and Security: Procedia Computer Science, no. 85, pp. 878–890 (2016)

[6] Kotsiantis, S.B., Zaharakis, I., Pintelas, P.: Supervised machine learning: a review of classification techniques. Emerging Artif. Intell. Appl. Comput. Eng. 160, 3–24 (2007)

[7] Shao, Z., Jin, X., Jiang, W., Chen, M., Chiang, M.: Intra-data-center traffic engineering with ensemble routing. In: INFOCOM, 2013 Proceedings IEEE, pp. 2148–2156. IEEE (2013)

[8] Vik, K.-H., Halvorsen, P., Griwodz, C.: Multicast tree diameter for dynamic distributed interactive applications. In: INFOCOM 2008. The 27th Conference on Computer Communications IEEE. IEEE (2008)

[9] Webb, S.D., Soh, S., Lau, W.: Enhanced mirrored servers for network games. In: Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games, pp. 117–122. ACM (2007)

[10] Guo, J., Liu, F., Zeng, D., Lui, J.C., Jin, H.: A cooperative game based allocation for sharing data center networks. In: INFOCOM, 2013 Proceedings IEEE, pp. 2139–2147. IEEE (2013)

[11] Xu, K., Zhang, Y., Shi, X., Wang, H., Wang, Y., Shen, M.: Online combinatorial double auction for mobile cloud computing markets. In: Performance Computing and Communications Conference (IPCCC), 2014 IEEE International, pp. 1–8. IEEE (2014)

[12] Seung, Y., Lam, T., Li, L.E., Woo, T.: Cloudflex: seamless scaling of enterprise applications into the cloud. In: INFOCOM, 2011 Proceedings IEEE, pp. 211–215. IEEE (2011)

[13] Zaki, Y., Chen, J., Potsch, T., Ahmad, T., Subramanian, L.: Dissecting web latency in ghana. In: Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 241–248. ACM (2014)

[14] Wang, H., Shea, R., Ma, X., Wang, F., Liu, J.: On design and performance of cloud-based distributed interactive applications. In:

2014 IEEE 22nd International Conference on Network Protocols (ICNP), pp. 37–46. IEEE (2014)

[15] Pujol, E., Richter, P., Chandrasekaran, B., Smaragdakis, G., Feldmann, A., Maggs, B.M., Ng, K.-C.: Back-office web traffic on the Internet. In: Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 257–270. ACM (2014)

[16] Alnuaimi, S. S., Sundararajan, E. A. & Abd, A. H. 2022. Data distribution optimization over multi-cloud storage. Journal of Theoretical and Applied Information Technology 100(5).(2022)