An Intent-based Framework for Vehicular Edge Computing

TianZhang He
Faculty of Information Technology
Monash University
Melbourne, Australia
0000-0002-5472-7681

Adel N. Toosi
Faculty of Information Technology
Monash University
Melbourne, Australia
adel.n.toosi@monash.edu

Negin Akbari
Faculty of Information Technology
Monash University
Melbourne, Australia
negin.akbari@monash.edu

Muhammed Tawfiqul Islam
School of Computing and Information Systems
The University of Melbourne
Parkville, Australia
tawfiqul.islam@unimelb.edu.au

Muhammad Aamir Cheema
Faculty of Information Technology
Monash University
Melbourne, Australia
aamir.cheema@monash.edu

***Abstract*—The rapid development of emerging vehicular edge computing (VEC) brings new opportunities and challenges for dynamic resource management. The increasing number of edge data centers, roadside units (RSUs), and network devices, however, makes resource management a complex task in VEC. On the other hand, the exponential growth of service applications and end-users makes corresponding QoS hard to maintain. Intent-Based Networking (IBN), based on Software-Defined Networking, was introduced to provide the ability to automatically handle and manage the networking requirements of different applications. Motivated by the IBN concept, in this paper, we propose a novel approach to jointly orchestrate networking and computing resources based on user requirements. The proposed solution constantly monitors user requirements and dynamically reconfigures the system to satisfy desired states of the application. We compared our proposed solution with the state-of-the-art networking embedding algorithms using real-world taxi GPS traces. Results show that our proposed method is significantly faster (up to 95%) and can improve resource utilization (up to 76%) and the acceptance ratio of computing and networking requests with various priorities (up to 71%). We also present a small-scale prototype of the proposed intent management framework to validate our solution.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The automotive industry is one of the fastest-growing industries. In recent years, the increased use of onboard microprocessors such as On-Board Units (OBUs) and sensors technology has led to technological advancements that enabled vehicles to provide various safety and driver assistance-related systems. For example, modern cars can autonomously drive to their destination, warn the driver of external hazards, and avoid collisions. However, many of these applications' growing demands for computational resources urge the use of the communication infrastructure and connection with Road Side Units (RSUs) to offload the heavy tasks. Moreover, vehicles are becoming increasingly connected, and V2X (Vehicle to Everything) communications enable vehicles to communicate with each other and the outside world, allowing applications to go beyond internal functions and provide improved awareness of impending events over a wider area. For many conventional connected vehicles, the network was only responsible for transferring data from vehicles to the cloud. However, applications are evolving into a highly distributed layer that resides directly within the network fabric. In fact, it is crucial to support many real-time applications and perform analytics at the edge as close as possible to the data source. Consequently, Vehicular Edge Computing (VEC) [1], [2] has become the mainstream paradigm to meet strict performance requirements such as response time and network bandwidth of real-time applications.

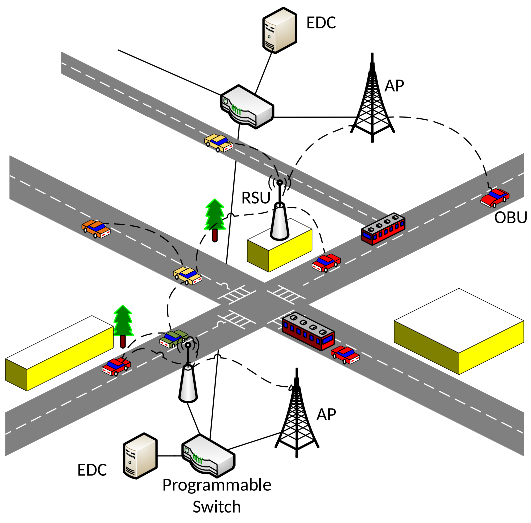


Fig. 1: Vehicular Edge Computing Overview

With an increasing number of computational nodes, for example, cloud, edge devices, vehicles, RSUs, and compute-

Identify applicable funding agency here. If none, delete this.

enabled network components, the networking operation domain in VEC is becoming more complex. Fig. 1 depicts one such VEC scenario where mobile vehicles may establish a network connection with other vehicles (inter-vehicle communication), RSUs, and access points (base stations). In addition, RSUs can be interconnected to other RSUs or to access points by either a wired or wireless network. Although each vehicle may have extra computing resources in its OBU, heavier computational workloads can be offloaded to the Edge data centers (EDC). Thus, it is challenging to efficiently orchestrate and manage the underlying networking and computing resources based on various service requirements in such a VEC environment. In other words, developing, deploying and operating applications in VEC environments is not trivial. Therefore, it is essential to create mechanisms to automatically capture the applications' deployment requirements (intents) to activate and assure them network-wide. Existing solutions try to address these issues mainly through service placement approaches and task offloading techniques without taking networks into consideration [3]. In this work, we aim to bridge the gap between the deployment requirements of VEC applications (business intent) and what the network delivers by building required algorithms considering both the computing and networking requirements of the applications.

Software-Defined Networking (SDN) is a technology that helps tackle the network management and orchestration complexity [4] and has been widely used in VEC and Mobile Edge Computing (MEC) environments [5], [6]. SDN centralizes the network's control plane and provides automation, cost-efficiency, programmability, and greater efficiency for network management. In recent years, Intent-Based Networking (IBN) based on SDN is also emerging to automate networks further. It provides network intelligence by evoking a high-level intent, detecting potential deviations from that intent, and prescribing actions required to ensure that the intent is always satisfied. Based on IBN techniques, the application provides intents to indicate the desired network requirements, such as network bandwidth and end-to-end delay. Several industrial vendors, such as Cisco and VMWare, also focus on the IBN agility for edge network management. Following the industry trend, in this work, we propose an intent-based manager to orchestrate both networking and computing resources for the VEC applications. Intent can be considered a high-level abstract declaration for applications describing their desired state or result. For example, a service provider may want an autonomous vehicle to maintain low-latency and high-throughput connections for its image processing service.

To the best of our knowledge, this is one of the earliest efforts to install and support joint networking and computing intents for VEC applications. Current Virtual Network Embedding (VNE) algorithms [7]–[9] are not suitable for the intent installation problem in the VEC environments for multiple reasons. Firstly, VNE algorithms do not allow allocations of multiple virtual nodes to the same compute node. In addition, traditional virtual network requests (VNR) are treated as standalone, while an intent of a VEC application may need to be compiled into multiple VNRs. Current VNE algorithms also do not support adding location constraints as needed by VEC intents. Thus, they are not capable of handling the mobility aspect of the users/applications. Apart from this, there is no computing requirement within the current intent framework and models. Last but not least, current VNE algorithms are not suitable for the problem of intent installation with priorities. Simply assigning priorities to the intents does not resolve the issue, as we also need to satisfy users' Quality of Service (QoS) requirements.

To address these problems, we propose an efficient online algorithm for intent installation with different priorities while considering both computing and networking-related properties. The key **contributions** of the paper are as follows:

- We introduce the computing resource and location requests into Intent-Based Networking for VEC.
- We propose a priority- and location-aware algorithms for the installation and management of intents with different priorities.
- We compare our proposed algorithm with the state-of-the-art VNE algorithms in a large-scale simulation with real-world data sets and edge networks.
- We implement and evaluate the proposed intent-based edge computing with a real SDN controller on a Mininet emulation platform.

II. System Overview

In this section, we first showcase the proposed intent management framework. Then we highlight the intent features available to the applications running in a VEC environment. Lastly, we discuss the intent life-cycle.

A. Intent Management Framework

Current intent-based networking frameworks only allow the mapping of application network resource requirements. In this paper, our goal is to extend the intent framework to allow the expression of both compute and network elements along with application QoS. Thus, we need to holistically manage the edge/cloud platform for compute resources, orchestrate the container placements for micro-services, and utilize the network controller to map the virtual network. As shown in Fig. 2, by integrating SDN controller (e.g. ONOS or OpenDayLight), edge and cloud platform (e.g. OpenStack), and container orchestrator (e.g. Kubernetes), we propose an intent-based manager to cohesively orchestrate both networking and computing resources.

The applications' intents are submitted to the intent-based manager using a declarative manifest. The manager then extracts the requirements of each intent to check whether the intent can be installed with the existing compute and network resource capacity. For a successful intent compilation, the extracted compute requirements from the intent are transformed into a resource allocation task with the help of the edge/cloud platform and the container orchestrator. The network request is also transformed into a VNR and the manager instructs the SDN controller to install the network intent.

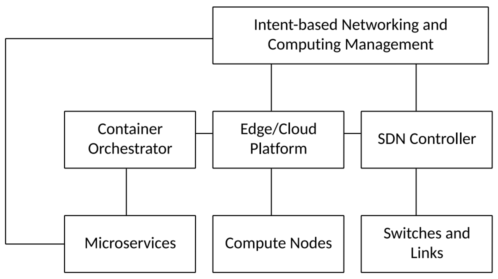


Fig. 2: Intent management framework

B. Intent Features

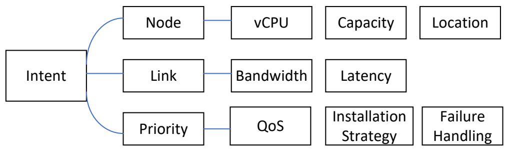


Fig. 3: Intent features and constraints

Fig. 3 shows the features available to the application to be expressed as per intent in the proposed intent management framework. We categorize the intent features in three groups as follows.

1) **Node:** An application/service can choose a set of nodes (locations) where the service can be executed (location constraints). In addition, the compute resource requirements (e.g., vCPU, memory or storage) for the service can also be specified (resource constraints).
2) **Link:** The application/service can choose the desired bandwidth requirement and express the minimum expected latency for the service to function correctly (network constraints).
3) **Priority:** As multiple services can be deployed across the system with competing and conflicting interests, intent from each service must have a priority which should be specified while submitting the intent. Thus, depending on the intent priority and the QoS requirements, proper intent installation strategy and intent failure handling mechanisms can be followed.

C. Intent Life-cycle

The proposed Intent Framework allows applications to specify both their network and compute resource requirements. The intent manager accepts the intent specifications and compiles them into installable intents that require some actions to meet the desired application state. Finally, when the actions are carried out in both the network and compute environments, some changes are made, for example, flow rules being pushed to switches or compute resources reserved to deploy a microservice on a node.

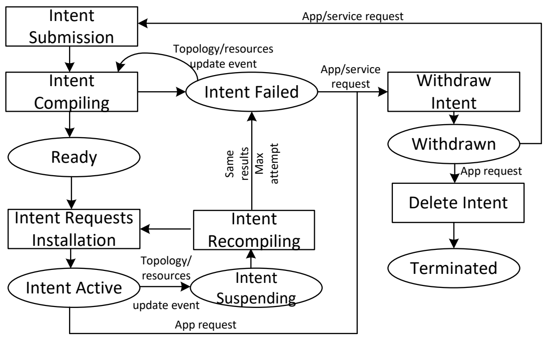


Fig. 4: Intent lifecycle

Fig. 4 depicts the complete life-cycle for the intents in the proposed framework. An intent can be in one of the following states: *Ready, Active, Suspending, Failed, Withdrawn, Terminated* (represented in oval shapes in the figure). The intent manager takes the actions to make changes to the environment and satisfy intent requirements. These actions are represented with rectangular shapes. We provide a brief overview of the key actions in the life-cycle of the intents.

- **Intent Submission:** Intents are submitted by the applications/service providers to the intent management framework. Upon receiving an intent asynchronously, the intent manager transforms the intent into several compute and network requests which can be used for the intent compilation phase.
- **Intent Compilation:** The resource discovery is made by the SDN controller and the VM/container orchestrator. The intent manager can communicate with them to get regular updates on network topology changes and the resource capacity of the compute nodes. Thus, if the network and compute resources are sufficient to handle the submitted intent, the intent will be compiled successfully, and the intent state becomes *Ready*. Otherwise, the intent state goes to the *Failed* state after exhausting the sufficient retry attempts.
- **Intent Installation:** A *Ready* intent can be installed in the system by reserving both the network and compute resources, such as compute and bandwidth for an application. When the intent is installed successfully, the intent state is changed to *Active*.
- **Intent Recompilation:** An *Active* intent might be suspended by the application or due to a topology and resource update. In this case, the intent state is *Suspending*, and the intent enters the Intent Recompilation phase. If the desired system state can be reached for the intent after a few recompilation attempts, then the intent again enters an *Installation* phase. Otherwise, the intent state is *Failed*.
- **Intent Withdrawal:** An application can request at any time to withdraw both an active or a failed intent. In this case, the system withdraws the intent and, if the intent is

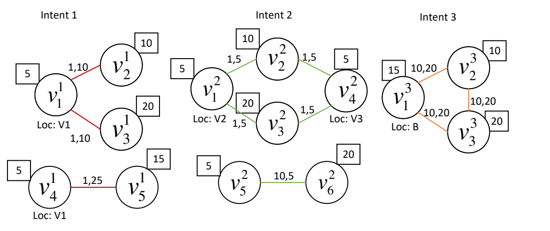


Fig. 5: An example of intents and compiled requests

active, takes back all the allocated network and compute resources. A *Withdrawn* intent can be resubmitted as a new intent based on the intent requirement template.

- **Intent Deletion:** While in the *Withdrawn* state, the application can request to delete the intent requirement template entirely from the system. At this point, the intent state will be *Terminated*.

III. System Model

The phases of intent installations and contention resolutions between the intents can be translated into an Online Virtual Network Mapping or Virtual Network Embedding (VNE) problem. The acceptance ratio and average resource utilization are critical parameters needed to be optimized.

Vehicular Edge Computing (VEC): VEC or a substrate network can be modeled as a weighted undirected graph $G(N, L, A_N, A_L)$ including the global network hardware, network links, edge devices and end users, where N denotes the set of physical nodes and L denotes the set of the physical network link. A_N and A_L denote attributes associated with nodes and links, respectively. For concise modeling, we consider CPU, memory capacities, and location constraints for node attributes, and bandwidth and latency for link attributes. We use P to denote all loop-free paths within the VEC network and P^k_{uv} to denote k shortest paths between node u and v.

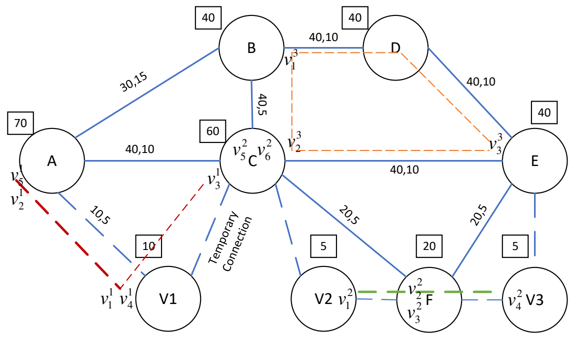


Fig. 6: An example of intent installation on substrate network

Intents and requests: Let I^m indicate the intent submitted by the application/service m. Let π^m denote the priority of Intent I^m. After the intent compiling process, I^m can be compiled into several virtual networking and computing requests R^m_i where i denotes the ith requests of intent I^m. As a microservice architecture, each compiled request of an intent can also be modeled as an undirected graph $R^m(N^m, L^m, C^m_N, C^m_L)$, where C^m_N and C^m_L denote the set of node and link constraints, such as the CPU, memory, and location constraints for virtual nodes, and bandwidth and latency for virtual links. For instance, in Fig. 5, *intent1* is compiled into two requests, where computing requirement of virtual node v^1_1 is 5. Bandwidth and delay requirement of the virtual link between v^1_1 and v^1_2 are 1 and 10, respectively. Virtual nodes v^1_1 and v^1_4 should be allocated in physical node $V1$, i.e., user node. As a result, if a mobile user reconnects to another EDC, virtual links associated with v^1_1 and v^1_4 need to be rerouted or other virtual nodes associated with these links might need to be relocated accordingly.

Location constraints $C^m_N(v) = n(v)$ of a virtual node v can be divided into *fixed* and *mobility-related* location constraints. For the fixed location constraint, the virtual node location constraint is associated with a stationary physical node, such as a certain roadside unit, gateway, or edge data center. However, if location constraints are associated with mobile end-users such as autonomous vehicles or pedestrians, the location of the virtual node can change over time. In other words, $n(v)$ is a mobile node. The intent-based orchestrator actively monitors and maintains the intent installation for mobility-related location constraints due to the current location of the virtual node.

Problem Description: The installation of a set of intents is defined by mappings: $M\{I^m\} : \{R^m(N^m, L^m, C^m_N, C^m_L) \rightarrow G(N, P^k_{uv}, A_N, A_L)\}$, from a set of R^m to G, where $N^m \subset N$ is the node mapping and $L^m \subset P^k_{uv}$ is the virtual link mapping to the network path. An intent installation is successful when its compute resource requirements and network resource requirements are all satisfied. Fig. 6 illustrates a possible mapping for all requests of the three intents in Fig. 5, installed on the substrate network.

Objectives: The objective of intent installation is to maximize the intent acceptance ratio owing to their priorities while efficiently utilizing both computing and networking resources. Let binary variable $X^{m,t} = \{1, 0\}$ denote whether intent I^m is successfully installed at time t or not and $X^{m,t}_i = \{1, 0\}$ denote whether request R^m_i of intent I^m can be satisfied or not. When intent I^m is not submitted, $X^{m,t} = 0$. Thus, the success of a general intent installation is defined as:

$$X^{m,t} = X^{m,t}_0 \wedge X^{m,t}_1 ... \quad (1)$$

Therefore, the long-term intent acceptance ratio can be presented as follows:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} \sum X^{m,t}}{\sum_{t=0}^{T} |\{I^{m,t}\}|} \quad (2)$$

where $|\{I^{m,t}\}|$ is the total number of submitted intents at t.

We define three different priority levels for intents: *high*, *mid* and *low*. With different priority levels, we define two installation semantics: 1) a high and low priority intent is successfully installed only if all its compiled requests are satisfied and embedded $\forall X_i^m = 1$; and 2) a median-priority intent allows its requests to be installed partially.

To quantify the acceptance ratio of mid-priority intents, we need to model the long-term request acceptance ratio, which can be formulated as:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} \sum X_i^{m,t}}{\sum_{t=0}^{T} \left|\left\{X_i^{m,t}\right\}\right|} \tag{3}$$

where $\left|\left\{X_i^{m,t}\right\}\right|$ is the total number of requests at time t.

To the VEC provider, the cost of intent I^m installation is modeled as the sum of total resource requirements:

$$\kappa^m = \alpha \sum_{n\in N^m} cpu_n + \beta \sum_{n\in N^m} mem_n + \gamma \sum_{l\in L^m} bw_l/delay_l \tag{4}$$

where α, β, and γ are weights for resources in different categories valued by the VEC provider. Thus, the revenue of intent I^m installation for VEC provider at time t can be formulated by:

$$\varepsilon^{m,t} = \sum X_i^{m,t} \cdot \kappa_i^{m,t} \tag{5}$$

Similar to [10], the revenue to cost ratio is used to quantify the long-term resource utilization:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} \varepsilon^{m,t}}{\sum_{t=0}^{T} \kappa^{m,t}} \tag{6}$$

It is known that the general VNE problem is NP hard [7]. Thus, we rely on heuristics to practically solve the problem.

IV. Online Intent Management

Current solutions and algorithms [10]–[12] of virtual embedding problem cannot be directly applied to the intent installation problem in VEC environments. In this section, we describe our proposed online intent management solution, which includes a priority-aware intent (PAI) installation algorithm and corresponding location-aware mapping (LAM) algorithm for intent-based vehicular edge computing. The intuition behind our proposed priority-aware intent-based processing algorithm is as follows:

- Microservices (requests) with location constraints should be processed first to increase the acceptance ratio.
- Microservices with less complexity and resource demands that have less impact on other intent installations will be processed first.
- To increase the acceptance ratio of higher-priority intents and total resource utilization, intent requests with higher priority will be installed first.
- If there is no request left for the highest installation level, we process compiled requests of all other intents.
- In the end, if there is no higher priority intent left for processing, we consider the requests with the lowest installation priority.
- There is a significant difference in processing and operation costs between intent reinstallation with virtual node relocation and merely remapping its virtual link. The mapping may change due to the user mobility after a virtual node has been allocated in the mobile user node. Therefore, allocating other nodes that share virtual links in proper locations can reduce maintenance costs for intent installation.

Algorithm 1: Priority-Aware Intent Installation (PAI)

```
Input  : $G$ edge network graph with previous mapping
         information $M(t')$
Input  : time interval t, $\{I_{submit}\}, \{I_{suspend}\}, \{I_{fail}\}$
1 foreach $I \in \{I_{suspend}\}$ do
2     if CheckLink(I) == Failed then
3         RemapPath(I);
4         if status(I) == Failed then
5             $I_{fail} \leftarrow I$
6 Compile($\{I_{submit}\} \cup \{I_{fail}\}$):
7 Sort($\{I^{high}\}$); $\forall I \in \{I^{high}\}$, InstallAll($I$);
8 Sort($\{R^{mid}\}$); $\forall R \in \{R^{mid}\}$, InstallBest($R$);
9 Sort($\{I^{low}\}$); $\forall I \in \{I^{low}\}$, InstallAll($I$);
```

A. Priority Aware Intent Installation

With different priorities, the intent contention resolution algorithm is the key component for IBN-based edge computing management and orchestration. At each time interval t, priority-aware installation algorithm first checks intents associated with *Suspending* event. If virtual link remapping cannot satisfy the intent, it changes the intent to *Failed* state for reinstallation. We set the retry threshold for reinstallation from *Failed* intent to *three*. To increase the acceptance ratio of intents with higher priorities and reserve sufficient resources for subsequent intents, we divide intents into two installation semantics and policies (Algorithm 1):

- *InstallAll(I)*: must satisfy all compiled requests of one intent.
- *InstallBest(R)*: satisfy as many compiled requests as possible.

For each priority group, the intents I^{high}, I^{low} and compiled requests R^{mid} are sorted in ascending order based on the cost model shown in Eq. (4).

B. Location-Aware Mapping

For intents and compiled requests embedding, we propose the location-aware microservice mapping (LAM) algorithm by considering location constraints (Algorithm 2). In addition to considering the computing and networking resource requirements of requests within intents, our proposed embedding algorithm also considers the location constraints, including fixed and mobility-related location constraints. Traditionally, location constraints of virtual network requests are fixed, such as Virtual Network Function (VNF) location constraints of Service Function Chaining (SFC) [13] and host constraints due to data security or privacy. For the intents with mobility-related location constraints, we allocate requests such that it minimizes the possibility of virtual node reallocation.

Algorithm 2: Location-Aware Mapping (LAM)

```
Input   : Requests $\{R\}$, Network $G$, Search depth $d$
Output  : $\{M(R)\}$ mappings of $\{R\}$
1  $\{R\} \leftarrow \{R^{loc}\}$; or $\{R\} \leftarrow \{R^{non}\}$;
2  sortRequests($\{R\}$);
3  foreach $R \in \{R\}$ do
4      foreach $v \in \{v^{loc}\}$ do
5          $M(R) \leftarrow$ mapNode($G, v, n(v)$);
6          if $M(R)^v$ == null then
7              status(R) = Failed; goto line 18;
8      sortNode($\{v^{non}\}$); Q.enqueue(getMinNode($\{v^{non}\}$));
9      while $Q \neq \emptyset$ do
10         $\{u\} \leftarrow Q$;
11         foreach $u \in \{u\}$ do
12             $M(R) \leftarrow$ mapNode($G, u, M(R)$);
13             if $M(R)^u == null$ then
14                 $M(R) \leftarrow$ mapNode($G, u, sortNode(N_u^+)$);
15             if $M(R)^u == null$ then
16                 status(R) == Failed; goto line 18;
17             $\{u\} \leftarrow$ Neighbors($R, u, M(R)$);
               Q.enqueue($\{u\}$);
18     if status(R) == Failed then
19         releaseMap(R);
```

Following our installation semantics, the compiled requests $\{R\} = I^m$ of a high-level or low-level priority intent, and all compiled requests with mid-level priority $\{R\} = \sum I^m$ are processed sequentially. LAM first allocates requests with fixed and mobility-related location constraints R^{loc}. Then, it allocates other requests without any location constraints R^{non}. For each successful node mapping *nodeMap*, the mapping result satisfies both node and link requirements. If a virtual link $l_{u,v}$ exists in R and virtual node v has been mapped to node $n(v)$, i.e., $M(R)^v = n(v)$, the virtual link is mapped based on k shortest path between the testing node n and $n(v)$. If any node mapping $M(R)^v$ is failed, the request mapping is also failed without further processing.

From steps 2 to 7, for each R^{loc}, virtual nodes $\{v^{loc}\}$ with fixed and mobility-related location constraints $\{n(v)\}$ are mapped firstly. From steps 8 to 17, the remaining virtual nodes are mapped in a breadth-first search manner. At step 8, virtual nodes are sorted based on scores calculated in Eq. (7); and the virtual node with minimum cost is selected (u) as the start node. The virtual node score in a request R is formulated as:

$$S(i')^R = (\alpha' \cdot cpu + \beta' \cdot ram) \cdot k_{nn,i'}^{bw} \tag{7}$$

where cpu and ram are normalized resource requirements of virtual node i'. The ratio of α' and β' is based on the computing resource requirement of the selected virtual node i', $\alpha'/\beta' = cpu^{i'}/ram^{i'}$. $k_{nn,i}^{bw}$ is the average neighbor degree with bandwidth as the weight. $k_{nn,i}^{bw} = \frac{1}{s_i}\sum_{j\in N(i)} bw_{ij}k_j$, where s_i is the weighted degree of node i, $N(i)$ is the set of node i's neighbors, k_j is the degree of node j which belongs to $N(i)$. bw_{ij} is the bandwidth of the edge (link) that connects node i and j.

The node mapping procedure continues until the searching queue Q is empty or a mapping is failed. For each node mapping, already selected physical nodes in the request mapping $M(R)$ are tested first to minimize the link mapping cost at step 12. If there is no matching, candidate physical nodes within the search depth are sorted (Eq. (9)) and tested for mapping at step 14. At step 17, unexplored adjacent virtual nodes are enqueued based on the virtual node sorting for further mapping. The candidates nodes N_u^+ for unmapped virtual node u are the intersection set of edge nodes within search depth $d_{u,v}$ (the number of switch hops) of the current mapping node $n(v) \in M(R)$,

$$N_u^+ = \bigcap_{v\in M(R)} N_{n(v)}^+(d_{u,v}) \tag{8}$$

where $N_{n(v)}^+(d_{u,v})$ is all the nodes within search range of node $n(v)$ and $n(v)$ is the mapping node of virtual node v that $n(v) : v \rightarrow n$. The search depth is calculated as $d_{u,v} = d \cdot \lceil Delay_l/\mu \rceil$, $l_{u,v} \in L_i^m$ where virtual link $l_{u,v}$ exists, μ is delay coefficient ($\mu = 10ms$) and d is the search range coefficient ($d = 2$). If there is no mapped node, $N_u^+ = G(N)$ for the first virtual node mapping. When the request mapping is successful, the remaining physical resources are updated accordingly.

At step 14, the candidate physical nodes N_u^+ are sorted in a descending order based on the node score model (Eq. 9). The score of edge node i in the substrate network is formulated as follows:

$$S(i)^{sub} = (\alpha \cdot cpu + \beta \cdot ram) \cdot k_{nn,i}^{bw} \tag{9}$$

where the coefficients of remaining cpu and ram resources are $\alpha + \beta = 1$, coefficients α and β are based on the remaining computing resources of all nodes within the search distance as:

$$\alpha/\beta = \sum_{j\in N_i^+(d)} cpu^j / \sum_{j\in N_i^+(d)} ram^j \tag{10}$$

Compared with traditional VNE solutions, we allow virtual nodes embedded in the same edge node to increase node utilization and reduce the network cost. In other words, each edge node is a cluster of connected physical hosts. Multiple virtual nodes of the same request can be allocated in the same edge data center and the same physical host. For the virtual link mapping, a remote edge node with longer paths results in reduced bandwidth resource for other intent installations. In other words, it may reduce the intent installation acceptance ratio when the network resources are limited. Furthermore, most mapping algorithms consider the node and link mapping separately. However, virtual links of microservices at the edge are distance and latency-sensitive. Selecting all node locations without considering delays of a path can dramatically increase the reject ratio of intent installation. Therefore, we introduce search depth and distance parameters in node sorting to allocate requests in proximity.

TABLE I: Physical and intents parameters

Physical Networks		Intents Parameters	
edge node	200	req num.	[1, 4]
edge link	758	vir. node/link	[2, 4]/[2, 4]
node CPU	[10,40]	vir. CPU	[1, 2]
node RAM	[10,80]	vir. RAM	[1, 4]
link bw	[400,1000]	bw/delay	[1,2]/[10,100]

V. Performance Evaluation

In this section, we compare the proposed intent installation algorithm with baseline algorithms based on the existing virtual network embedding algorithms, *grcrank* (Global Resource Capacity) [11], *rwrank* (Markov Random Walk PageRank) [10], and *nrmrank* (Node Ranking Metric) [12] in terms of intent acceptance ratio, resource utilization, and execution time. For a large-scale simulation, we utilize the real-world taxi GPS dataset in Shanghai (April 1, 2018)[1] and the locations of base stations from Shanghai Telecom[2].

A. Experiment Configurations

We extract the taxi GPS data within one hour with the number of taxis ranging from 1000 to 3000. The location of each edge node or server is calculated based on the density of base stations of Shanghai telecom by the K-mean algorithm (K=200) [14]. Physical links within the VEC network are generated based on Delaunay triangulation algorithm [15]. As a result, there are 758 physical links with a 6.6852 km average distance between edge servers. The average distance between a user and the nearest edge station is 1.63 km.

The delays of wireless connection between users and base stations and wired network link are calculated based on the following model:

$$t = t_{wireless} + t_{wired} \tag{11}$$

$t_{wireless} = W * \log_2 \frac{Sg_t}{N}$ where the channel gain g_t is $127 + 30 * \log(d)$, where d is the distance between the user and local base station [16]. The channel bandwidth W is set to 20 Mhz, the noise power N is $2 * 10^{-13}$ Watt, and the wireless transmit power of vehicle S is 0.5 Watt. The propagation time of wired links in milliseconds is calculated as $t_{wired} = 0.005 * d$, where d in km is the length of the direct optical cables.

Fig. 7 depicts the number of intent submissions and suspend events over time, and the total submission and suspend event number of requests and intents with various user numbers. As you can see in the figure, we have a large number intents submitted by users before time 25. Table I illustrates the details of the physical network and intent parameters. Experimental results are reported based on the average of 10 randomly generated values.

B. Results Analysis

We evaluate the proposed PAI and LAM algorithms (*pailam*) described in Sections IV-A and IV-B in various aspects, namely intent acceptance ratio, resource utilization, and execution time.

Figure 8 illustrates the performance comparisons in terms of acceptance ratio, utilization ratio, and execution time with a various numbers of users. The ratio of location constraint requests to all requests is 0.1. It shows that our proposed LAM algorithm can efficiently install delay-sensitive requests with and without location constraints. LAM significantly increases the intent acceptance ratio (Fig. 8a) and utilization (Fig. 8b) by up to 58%-71% and 66%-76%, respectively, compared with other online installation algorithms. By considering the entire physical network for mapping, the execution time of baseline VNE algorithms is too high to suit the large-scale intent installation (Fig. 8d). Due to the appropriate candidate selections, the processing time of LAM is decreased by up to 95% compared with other online algorithms.

We further evaluate our proposed intent framework and *pailam* algorithm with various parameters, including the intent priority, search depth for the node mapping candidates, and the ratio of requests with location constraints (Fig. 9).

Intent priority: Figure 9a shows the reject ratio along the time, in which high, mid, and low-level intent priorities are evenly generated. Compared to the mid-priority intents, the reject ratio of high-priority intents are significantly smaller. Both mid and high-priority intents can be maintained at a high acceptance ratio (0.923-0.979 and 0.931-0.979). Intents with low priority (0.741-0.907) are rejected when one of the requests is not satisfied to reserve resources for higher-priority intents.

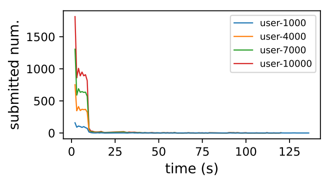
Allocation search depth: With fixed location constraints ratio of 0.1, we examine the acceptance ratio of *pailam* with various search depths for node mapping (Fig. 9b). The difference in acceptance ratio between depths 2, 4, and 8 is insignificant among different number of users. With a larger group of candidates, the depth d=4 has the best performance in all scenarios. However, the increase in algorithm execution time is considerably greater than the increase in performance compared to d=2. When d=8, the acceptance ratio decreases slightly because the larger group of candidates may lead to more mapping rejections due to the nature of submitted delay-sensitive intents.

Location constraints: Figure 9c illustrates the acceptance ratio when the percentage of user location-related requests varies for 2000 users. *pailam* uniformly and significantly outperforms other online algorithms in all scenarios. The acceptance ratio is the highest in the scenario where 10% of total intents are location-related (pailam-0.1). The acceptance ratio of location-related requests may be higher than the non-location-related requests when the computing and networking resources of $n(v)$ locations with mapping constraints are sufficiently large. For example, when the user's on-board processing resources and its network connections to edge servers are sufficiently large (0.5), the acceptance ratio is higher than 0.2 scenario.

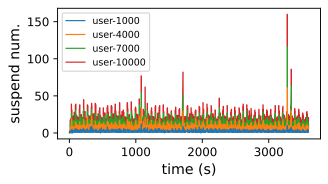
Link remapping and intent reinstallation: Suspended events are triggered along the time due to user mobility

[1]http://soda.shdataic.org.cn/download/31

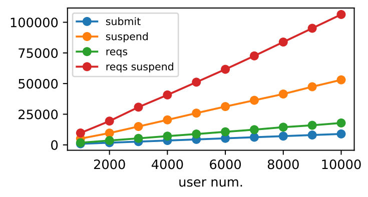
[2]http://sguangwang.com/TelecomDataset.html



(a) submitted num. of intents along time

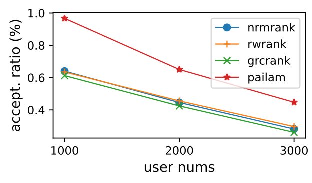


(b) suspended num. of intents along time

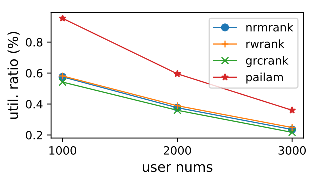


(c) total num. of submitted and suspended intents and requests

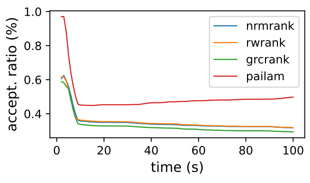
Fig. 7: Intent data statistics with various amount of users



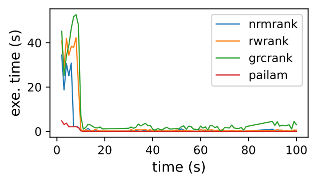
(a) acceptance ratio with various user amounts



(b) utilization ratio with various user amounts

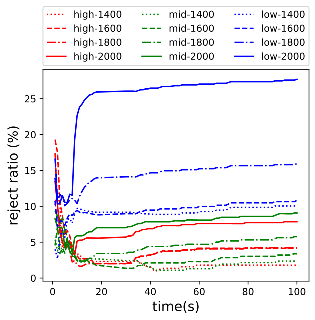


(c) acceptance ratio with 3000 users along time

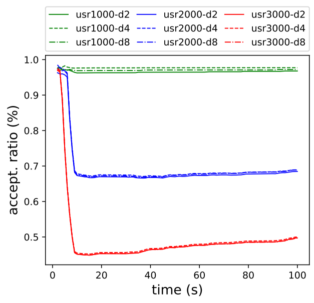


(d) execution time with 3000 users along time

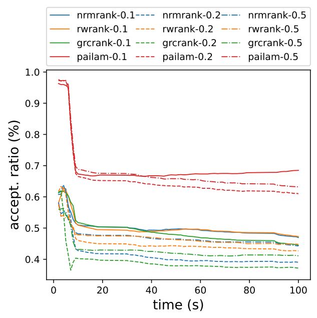
Fig. 8: Installation performance comparisons with different online algorithms



(a) accept. ratio with various priorities



(b) accept. ratio with various search depths



(c) user-location constraints with 2000 users

Fig. 9: Installation performance with various amount of users in priority, search depth, user location constraints

(Fig. 7b). Compared to baseline VNE algorithms, the proposed algorithm does not directly map the suspended requests as if they are new submissions. As a result, it largely reduces the execution time and the cost of VM or container migrations for intent reinstallation. This is because the operating cost of path rerouting is significantly smaller than live migration.

VI. Intent-Based Computing Prototype

In this section, we showcase an Intent-Based Vehicular Edge Computing prototype based on the SDN controller (ONOS) and Mininet Emulation platform.[3] Virtual node (VM and container) mapping and virtual link embedding are controlled by the intent-based framework. We generated a series of events to validate the feasibility, availability, and flexibility of our proposed system (Fig. 10). With a 100 priority (low-level), $Intent1$ is compiled into one request with v_1^1, v_2^1 and v_3^1 and links v_1^1-v_2^1 and v_1^1-v_3^1. Requirement of each node is 2 vCPUs and 4 GB RAM and location constraint of v_1^1 is $user1$. Each virtual link requires 20 Mbps of bandwidth and 30 ms of latency. With a 200 priority (mid-level), $Intent2$ is compiled into two requests with virtual link v_1^2-v_2^2 and v_1^2-v_3^2, respectively. The requirement of each node is 2 vCPUs and 4 GB RAM and the location constraint of v_2^2 is $EDC1$. The link requirement is 20 Mbps and 100 ms.

At time *i1* and *i2*, *intent1* and *intent2* are submitted and installed, respectively. At *e1*, $user1$ who is connected to EDC1 moves to a new position and gets connected to EDC4. As a result, a mobility event is raised to check the installation of *intent1* (Fig. 10(a)). At *e2*, EDC2 goes down. v_2^1 and v_3^1 are reallocated to satisfy *intent1* (Fig. 10(b)). At time *e3*, the link between S3 and S4 goes down (Fig. 10(c)). Between *e4* and *e5*, request v_1^2-v_3^2 of *intent2* failed due to EDC1 is down and v_3^2 location constraint cannot be satisfied (Fig. 10(d)). At $e5$, EDC1 is up again and request v_1^2-v_3^2 is reinstalled. As shown in Fig. 11 and Fig. 12, the prototype application can react to the

[3] You can find a demo at https://youtu.be/ZXBXdxug_x4

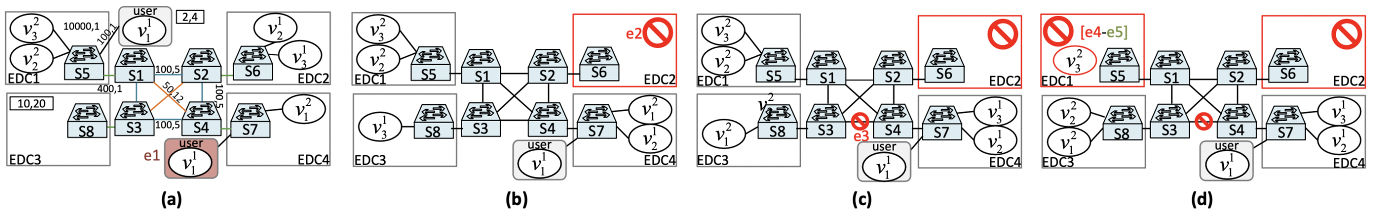


Fig. 10: Emulation scenarios. The numbers over the links show the bandwidth and delay, i.e. 400,1.

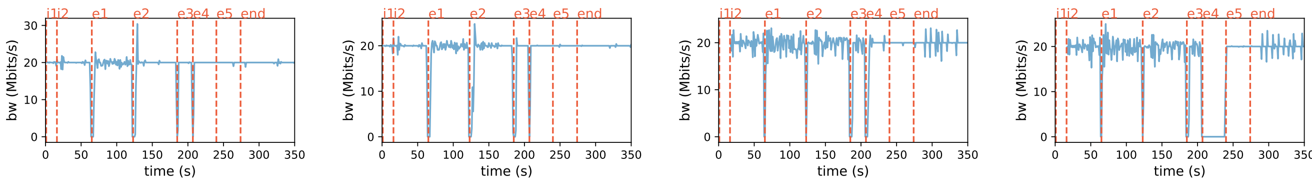

(a) v_1^1 and v_2^1 (b) v_1^1 and v_3^1 (c) v_1^2 and v_2^2 (d) v_1^2 and v_3^2

Fig. 11: Emulation performance in bandwidth along the time

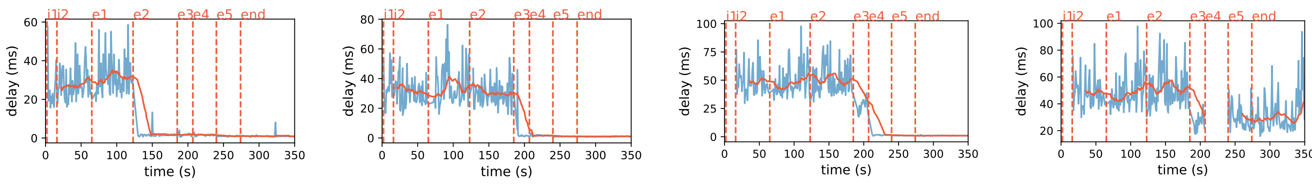

(a) v_1^1 and v_2^1 (b) v_1^1 and v_3^1 (c) v_1^2 and v_2^2 (d) v_1^2 and v_3^2

Fig. 12: Emulation performance in delay and its rolling average along the time

computing, networking, and mobility events, satisfy the intents requirements, and manage the life-cycle of intents efficiently.

VII. Related work

Virtual network embedding (VNE) [17] refers to the embedding of a virtual network in a substrate network. As the virtual resources are dynamically mapped onto the physical hardware, the exiting hardware utilization can be maximized. To provide custom user-defined end-to-end guaranteed services to end users, there are different problems that are addressed in this problem domain, such as optimal resource allocation, self-configuration and organization of the network. Yu *et al.* [7] showed an approach which combines path splitting, path migration, and customized embedding algorithms to enable a substrate network to satisfy a larger mix of virtual networks. Dietrich *et al.* [18] addressed the problem of multi-provider VNE with limited information disclosure (LID). EPVNE [19] is a heuristic algorithm that reduces the cost of embedding the Virtual Network (VN) request and increase the VN request acceptance ratio. Hejja and Hesselbach [20] proposed an online power aware algorithm to solve the VNE problem using less resources and less power consumption with end-to-end delay as a constraint. Jinke *et al.* [21] proposed a VNE model, where the high priority users can get extra resources compared to low priority users. Ogino *et al.* [22] proposed a VNE method to minimize the total substrate resources required during substrate resource sharing among multiple priority classes. Nguyen *et al.* [23] proposed a node-ranking approach, and a parallel GA-based algorithm for link mapping stage to solve online VNE problem. DeepViNE [24] is an Reinforcement Learning (RL)-based VNE solution, which automates the selection of problem features required in the DRL approach. MUVINE [9] is also an RL-based approach to be used as a prediction model for multi-stage VNE among the cloud datacenters.

Recently, there has been some research to improve the network functionalities with the help of SDN intents in different application scenarios. ONOS Intent framework [25] indicates the intent-based networking operations used in ONOS SDN controller. Han *et al.* [26] proposed an intent-based virtual network management platform based on software-defined NV to automate the management and configuration of virtual networks. Cerroni *et al.* [27] proposed a reference architecture and an intent-based North Bound Interface (NBI) for end-to-end service orchestration across multiple technological domains, with a primary use-case being the infrastructure deployment on the Internet of Things (IoT). DISMI [28] is also proposed as an intent-based north-bound interface of a network controller. Addad *et al.* [29] benchmarked the ONOS intent northbound interface using a methodology that takes into consideration the interface access method, type of intent and number of installed intents. OSDF [30] is an SDN-based network programming framework that provides high-level APIs to be used by managers and network administrators to express network requirements for applications and policies for multiple domains. Sanvito *et al.* [31] extended the intent Framework to make it able to both compile multiple intents together and to re-optimize their paths according to the network state based on flow statistics. [32] proposed architecture for automatic intent-based provisioning of secure services in multi-layer IP, Ethernet, and optical networks while choosing the appropriate encryption layer. [33] enables networks to make effective resource utilization and minimize the maximum link capacity utilization by using intent-based networking.

In summary, the existing VNE algorithms cannot place multiple virtual nodes to the same compute node. Furthermore, VNRs are processed as individual requests. However, our work supports VEC applications that need to be compiled into multiple VNRs. Our approach also supports adding location constraints for intents, and is capable of handling the mobility aspect of the users/applications. Unlike existing VNE algorithms, our approach is suitable for intent installation with priorities. Lastly, our approach incorporates computing requirements within current intent framework and models, while satisfying users' QoS requirements.

VIII. Conclusions and future work

In this paper, we proposed a novel intent framework to jointly orchestrate networking and computing requirements of applications based on user requirements in vehicular edge computing environments. The proposed solution constantly monitors user requirements and dynamically reconfigures the system to satisfy the desired states of applications. It can also efficiently optimize resource utilization and the acceptance ratio of computing and networking requests with various priorities. Results show that our proposed framework outperforms the state-of-the-art virtual network embedding algorithms in terms of acceptance ratio, resource utilization, and execution time. We also provided a small-scale prototype to validate our proposed framework. In the future work, we intend to fully implement our framework by extending the intent framework of the ONOS SDN controller. There are inherited periodic patterns of service mobility in vehicular edge environments. Therefore, we plan to take mobility patterns into consideration for intent re-installation and intent management.

References

[1] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular Edge Computing and Networking: A Survey," *Mobile Networks and Applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[2] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–19, feb 2019.

[3] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verı́ssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] L. Nkenyereye, L. Nkenyereye, S. M. R. Islam, C. A. Kerrache, M. Abdullah-Al-Wadud, and A. Alamri, "Software defined network-based multi-access edge framework for vehicular networks," *IEEE Access*, vol. 8, pp. 4220–4234, 2020.

[6] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *Proceedings of 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1202–1207.

[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 17–29, 3 2008.

[8] A. Razzaq, M. Hidell, and P. Sjödin, "Virtual network embedding: a hybrid vertex mapping solution for dynamic resource allocation," *Journal of Electrical and Computer Engineering*, vol. 2012, p. Article 5, 2012.

[9] H. K. Thakkar, C. K. Dehury, and P. K. Sahoo, "Muvine: Multi-stage virtual network embedding in cloud data centers using reinforcement learning-based predictions," *IEEE Journal on Selected Areas in Communications*, vol. 38, pp. 1058–1074, 6 2020.

[10] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, p. 38–47, apr 2011.

[11] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proceedings of IEEE INFOCOM 2014*, 2014, pp. 1–9.

[12] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3298–3304, 2018.

[13] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proceedings of 2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015, pp. 171–177.

[14] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, vol. 50, no. 5, pp. 489–502, 2020.

[15] P. Virtanen, R. Gommers, T. E. Oliphant, and et al, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[16] C. Niu, Y. Li, R. Q. Hu, and F. Ye, "Fast and efficient radio resource allocation in dynamic ultra-dense heterogeneous networks," *IEEE Access*, vol. 5, pp. 1911–1924, 2017.

[17] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[18] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-domain virtual network embedding with limited information disclosure," in *Proceedings of 2013 IFIP Networking Conference*, 2013, pp. 1–9.

[19] Y. Li and Y. Zhang, "EPVNE: An Efficient Parallelizable Virtual Network Embedding Algorithm," *Wireless Communications and Mobile Computing*, vol. 2019, p. 8416592, 2019.

[20] K. Hejja and X. Hesselbach, "Online power aware coordinated virtual network embedding with 5G delay constraint," *Journal of Network and Computer Applications*, vol. 124, pp. 121–136, 2018.

[21] C. Jinke, N. Xiumei, G. Huaxi, and Z. Linjie, "A user priority-based virtual network embedding model and its implementation," in *Proceedings of 2013 IEEE 4th International Conference on Electronics Information and Emergency Communication*, 2013, pp. 33–36.

[22] N. Ogino, T. Kitahara, S. Arakawa, and M. Murata, "Virtual network embedding with multiple priority classes sharing substrate resources," *Computer Networks*, vol. 112, no. C, pp. 52–66, 2017.

[23] K. Nguyen, Q. Lu, and C. Huang, "Efficient Virtual Network Embedding with Node Ranking and Intelligent Link Mapping," in *Proceedings of 2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, pp. 1–5.

[24] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual Network Embedding with Deep Reinforcement Learning," in *Proceedings of IEEE INFOCOM 2019 - INFOCOM WKSHPS*, 2019, pp. 879–885.

[25] A. Campanella, "Intent based network operations," in *Proceedings of 2019 Optical Fiber Communications Conference and Exhibition (OFC)*, 2019, pp. 1–3.

[26] Y. Han, J. Li, D. Hoang, J. H. Yoo, and J. W. K. Hong, "An intent-based network virtualization platform for sdn," in *Proceedings of 2016 12th*

International Conference on Network and Service Management (CNSM). IEEE, 1 2017, pp. 353–358.

[27] W. Cerroni, C. Buratti, S. Cerboni, G. Davoli, C. Contoli, F. Foresta, F. Callegati, and R. Verdone, “Intent-based management and orchestration of heterogeneous openflow/iot sdn domains,” in *Proceedings of 2017 IEEE Conference on Network Softwarization*. IEEE, 8 2017.

[28] P. Skoldstrom, S. Junique, A. Ghafoor, A. Marsico, and D. Siracusa, “Dismi - an intent interface for application-centric transport network services,” in *Proceedings of 2017 19th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 7 2017, pp. 1–4.

[29] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, H. Flinck, and M. Namane, “Benchmarking the onos intent interfaces to ease 5g service management,” in *Proceedings of 2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 12 2018, pp. 1–6.

[30] D. Comer and A. Rastegatnia, “Osdf: An intent-based software defined network programming framework,” in *Proceedings of 2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. IEEE, 10 2018, pp. 527–535.

[31] D. Sanvito, D. Moro, M. Gulli, I. Filippini, A. Capone, and A. Campanella, “Onos intent monitor and reroute service: Enabling plugplay routing logic,” in *Proceedings of 2018 4th IEEE NetSoft*. IEEE, 6 2018, pp. 456–461.

[32] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, “Automatic intent-based secure service creation through a multilayer sdn network orchestration,” *Journal of Optical Communications and Networking*, vol. 10, p. 289, 4 2018.

[33] A. Rafiq, M. Afaq, and W.-C. Song, “Intent-based networking with proactive load distribution in data center using ibn manager and smart path manager,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, pp. 4855–4872, 11 2020.