sensors

MDPI

Article

Advanced Computation Capacity Modeling for Delay-Constrained Placement of IoT Services

Balázs Németh * and Balázs Sonkoly

MTA-BME Network Softwarization Research Group, Budapest University of Technology and Economics, 1111 Budapest, Hungary; sonkoly@tmit.bme.hu

* Correspondence: balazs.nemeth@tmit.bme.hu

Received: 22 June 2020; Accepted: 6 July 2020; Published: 9 July 2020

Abstract: A vast range of sensors gather data about our environment, industries and homes. The great profit hidden in this data can only be exploited if it is integrated with relevant services for analysis and usage. A core concept of the Internet of Things targets this business opportunity through various applications. The virtualized and software-controlled 5G networks are expected to achieve the scale and dynamicity of communication networks required by Internet of Things (IoT). As the computation and communication infrastructure rapidly evolves, the corresponding substrate models of service placement algorithms lag behind, failing to appropriately describe resource abstraction and dynamic features. Our paper provides an extension to existing IoT service placement algorithms to enable them to keep up with the latest infrastructure evolution, while maintaining their existing attributes, such as end-to-end delay constraints and the cost minimization objective. We complement our recent work on 5G service placement algorithms by theoretical foundation for resource abstraction, elasticity and delay constraint. We propose efficient solutions for the problems of aggregating computation resource capacities and behavior prediction of dynamic Kubernetes infrastructure in a delay-constrained service embedding framework. Our results are supported by mathematical theorems whose proofs are presented in detail.

Keywords: network abstraction; virtual network embedding; 5G infrastructure; IoT services; Kubernetes

1. Introduction

Digital applications relying on information gathered by sensors and transmitted by infocommunication networks are becoming pervasive across industries. Data are continuously collected from production lines in mass-producing factories, while warehousing robots, delivery drones and driverless vehicles are ensuring the autonomous, cost efficient logistics of getting products to the end-customers. Agricultural sensors are providing unprecedented insight into the efficiency of industrial-scale food and crops production. A wide range of smart city sensors and actuators for traffic control, parking optimization, public transportation and sharing economy based urban mobility solutions make metropolitan lives more convenient and environment friendly. Autonomous video surveillance, consumer intelligence based advertisements, smart home appliances and smart phones make our lives safer, comfortable and filled with personalized user experience. Enormous amount of information and optimization opportunities lurk in this massive, hyper-connected system. The value of this huge amount of data can only be exploited if it is collected, analyzed, understood and acted upon. That is the core process to monetize information collected by various types of sensors, and that is the essence of the Internet of Things (IoT).

The high service quality requirements driven by consumer demands of the information society, pushes communication technology innovation and development forward. New applications built on top of the vast amount of process intelligence are coming out rapidly, and innovation is ongoing with a never-seen-before pace. These demands of the current world characterize the use-cases of 5G networks and their functional requirements, such as massive Machine Type Communication (mMTC), Ultra-Reliable Low-Latency Communication (URLLC) and enhanced Mobile Broadband (eMBB) [1]. A recent surge of 5G prototype architectures and the ongoing competition for commercial 5G network deployments are academia's and industry's response to these challenging requirements. 5G systems utilize the paradigms of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) to decouple the services' business logic from the hardware, and realize an efficient management of computation and communication resources [2].

A central optimization problem in realizing IoT applications using the enabling 5G technology, is allocating service components to resources, which is formalized in multiple NP-hard variants of the service placement problem. Service placement is one of the most difficult problems, as no solution exists which is scalable (i.e., do not rely on directly solving integer programs), provably meets all constraints and provides performance guarantees [3,4]. Due to the immediate relevance of the service placement problem, a wide range of practical heuristic algorithms have emerged, which are easy-to-deploy, highly extensible and possible to be continuously developed in an actual 5G software architecture [5].

In parallel to algorithmic research on the service placement problem, the computation infrastructure is also evolving, becoming more dynamic and heterogeneous to contribute to meeting the challenging, service-level user requirements. In contrast, the infrastructure model of placement algorithms mostly consider only fixed substrate capacities and rarely goes beyond different capacity volumes to model heterogeneity. In addition to tackling heterogeneity and dynamicity, infrastructure modeling is an essential tool to handle the difficulty of the VNE problem by devising higher abstraction resource views for better algorithmic scalabilty.

In this paper, we contribute to 5G-enabled IoT service placement systems by proposing theoretically founded infrastructure abstractions for aggregating capacities and modeling elastic resources. We propose a service component placement heuristic algorithm design schema for meeting constraints on service path delay requirements. We show how these results can be integrated into proven prototype architectures. The focus of our paper is modeling the modern capabilities of distributed computing infrastructures, where services are deployed with respecting end-to-end requirements. More specifically, our contribution is threefold:

- We propose a method to design the step-by-step delay bounds of a greedy service placement algorithm to meet end-to-end delay requirements.
- We define capacity aggregation to efficiently abstract the network resources, targeting scalability and information hiding. We propose an approximation algorithm to create the aggregate infrastructure view with the desired abstraction level.
- We propose an elastic resource model and admission control for the Kubernetes container management system's Horizontal Pod Autoscaler feature [6,7], and show how it can be integrated with service placement algorithms.

All of our results are complemented with rigorous mathematical proofs, shown in the Appendixes A–C .

The paper is structured as follows. Section 2 puts our work in the context of a concrete IoT service deployment, and reviews related work on network abstraction and service placement. Section 3 explains the base system model, and introduces the service placement problem variant solved by our earlier works. Section 4 presents our results on end-to-end delay constraints and integrates it into our service placement framework. Our theoretical results on infrastructure modeling are presented in Sections 5 and 6, which incrementally extend our base system model with capacity aggregation capability and resource elasticity modeling. Integration of our resource models to 5G-enabeld IoT orchestration systems is presented in Section 7. Finally, Section 8 concludes the paper.

2. Background and Related Work

2.1. Background

Meeting the Internet of Things communication requirements such as connectivity of vast number of devices with high mobility, low latency and high reliability bring grand challenges to communication networks. These are addressed by the 5G mobile technology, which enables the monetization of various sensor-aided IoT services. The enabling relationship between 5G and IoT is examined in [1], where the state-of-the-art is extensively surveyed, gaps and research challenges are identified. The survey studies Mobile Edge Computing (MEC), data center performance optimization, edge cloud and virtualization technologies to identify their role in realizing 5G-enabled IoT applications, such as cloud robotics, industrial cyber physical systems, smart vehicles and tactile internet. Facilitating the interactions between various entities of the networking ecosystem is required to realize end-to-end IoT services. Authors of [8] present their prototype for a functional 5G architecture for optimizing end-to-end IoT network slices, which is integrated with the Kubernetes-based Google Compute Engine.

In the following we present a remote security camera analytics and alerts use case, where the computation intensive and delay sensitive IoT services are deployed across big geographic distances by an Over-the-Top Service Provider (OTT SP). Figure 1 shows a wide area network from the Internet Topology Zoo [9], where a Communication Service Provider (CSP) gives network connection for an OTT SP between its service sites. The location of the security cameras is to the west (loi_w), while the end-user location (sap_e) is to the east. Service Access Points (SAPs) are the user locations where the service is to be consumed, possibly by another service or a person. Location of Interests (LOIs) represent location-bound points, where the sensors (i.e., security cameras) are located. These should be used as service delay reference points, because the service quality is primarily experienced here. All auxiliary functions realize the service needs to be allocated on a network path between the reference points with meeting the strict delay requirement. The CSP provides communication capabilities between the distributed cloud of a Cloud Provider (CP) having its computation nodes apart with significant geographic distance. The OTT SP is in a business relationship with a CP, reaching its data center through the CSP's network, while also owning a private cloud proximate to site loi_w.

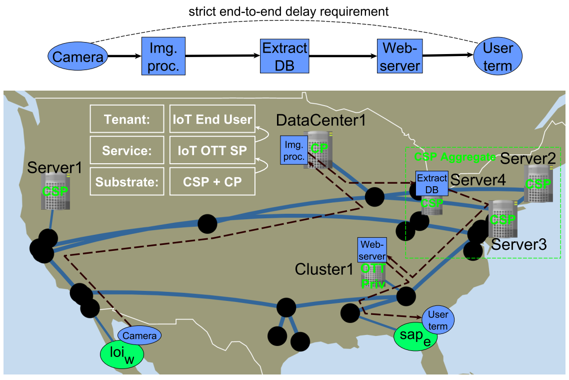

Figure 1. Demonstration of a wide area network with heterogeneous computation components with multiple stakeholders: Communication Service Provider (CSP) and Cloud Provider (CP) selling infrastructure to Over-the-Top Service Provider (OTT SP), who provide remote security camera analytics and alerts Internet of Things (IoT) services to its tenants.

If the OTT SP wishes to provide high quality IoT service over the rented infrastructure, it needs to know information about the CSPs network in significant detail to make informed decision about the IoT service component placements. On the other hand, the CSP may want to hide information from its tenants as they might be valuable or sensitive [10], where appropriate network and computation resource abstractions are essential. In addition, the CSP might want to optimize the running services' resource allocation to achieve higher computation resource efficiency, which should not be perceived by the OTT SP on its rented infrastructure view. The OTT SP's orchestration system needs to work on an appropriate infrastructure view which resolves the mentioned conflicts. Aggregating servers into higher abstraction nodes (as shown in CSP Aggregate node of Figure 1) also helps the embedding algorithm to scale up to carrier-grade network sizes. Finally, the Cloud Provider should also be able to give a descriptive view of its Kubernetes-based, optimized, dynamic data center resources.

2.2. Network Abstraction

Network infrastructure abstraction has always been important in modern network management to address the various application needs serving on top of the increasingly complex communication substrate. In a simple case, when only throughput requirements between endpoints need to be satisfied, the hose model has been proposed to describe Virtual Private Networks (VPNs) over a common substrate. The hose model has proved to be useful to efficiently provision the VPNs using a convenient tree structure [11]. In addition to meeting the bandwidth requirements, routing costs must be considered in practical cases. Such a system has been described by an abstracted network map [12], where the significant nodes are connected by cheapest, i.e., shortest in terms of cost metric, and highest bandwidth capacity paths, i.e., widest paths in terms of throughput. As virtual networks are the most valuable when they connect geographically distant locations, the underlying network infrastructure is likely to be managed by multiple providers. Virtual Network Embedding (VNE) is studied across multiple administrative domains in [10], where the authors devise a distributed algorithm, segmenting the virtual networks and embedding their routes in individual steps. The presented model lacks admission control capabilities on the domain capacity resources, which is one of the main contribution of our work.

As Software Defined Networking (SDN) enabled the network-wide control from a logically centralized controller, end-to-end network management based on high abstraction policies is convenient. SDN control architecture proves to be indispensable to implement IoT networks, furthermore, their appropriate performance requires distributed SDN controller placement [13], where abstracting the low-level management of individual devices is important. Network infrastructure abstraction is a key component to make the SDN concepts reach their full potential as studied in an SDN scalability survey [14]. Addressing these issues, the Big Switch model has been proposed to describe a network by its transport characteristics between the endpoints [15], which can be used to provision user traffic on top of the network, without studying its low-level details. Thanks to Network Function Virtualization (NFV), network services are decoupled from specialized hardware, packaged using virtualization technologies (such as containers and micro-services) and freely moved across computation locations [16]. Our recent works have generalized the Big Switch model to include computation capabilities of a network and keep up infrastructure modeling with the increasingly interleaving computation and communication resources. We have defined the Big Switch-Big Software (BiS-BiS) model [17], which has been used to build a multi-domain SDN/NFV-aware orchestration system [18,19]. These earlier versions of the BiS-BiS model naively sum the computation capacities and aggregate the functional capabilities of the hidden infrastructure, which ignores essential bottlenecks and lacks configuration options for the abstraction-performance trade-off. Emerging deployments of the NFV/SDN-enabled technologies in Mobile Edge Computing (MEC), Fog Computing, Industry 4.0, Internet of Things (IoT) and 5G make the networks increasingly heterogeneous. Thus, the naive BiS-BiS model's shortcomings are crucial in real deployments. Our current work addresses mathematically founded description of abstract, highly distributed, dynamic compute and network infrastructures.

2.3. Service Placement

The central problem in SDN and NFV-based computation network architectures is the service component placement problem, which has been well-studied by both industry and academia, stated in many forms. The general problem of service placement benefits from multiple areas of network management research, such as the Virtual Network Embedding (VNE) problem [20], Virtual Network Function (VNF) placement or Service Function Chain (SFC) resource allocation problem. Relations between these problem variants, and mutually useful results have been studied by survey papers [21]. The core of the service component placement problem addresses how to place the service-composing VNFs and how to route their connections with various constraints on the communication and computation characteristics. The problem is notoriously NP-hard and very difficult to approximate [3]. Moreover, modeling dynamic infrastructure capabilities such as reactive, automatic scale out of service components managed by modern container management systems is challenging. These behaviors cannot be formalized using linear constraints for integer programs, making solver-based optimization approaches hopeless [22]. A wide range of heuristic approaches have been proposed to solve many variants of the service placement problem, which are much more promising to meet high quality service requirements scaling to the immense size of IoT networks. We target complementing heuristic placement algorithms by providing theoretically founded generalizations of the infrastructure models to describe elasticity and abstraction. Indicating the hardness of the service placement related problems, no scalable solutions have been proposed so far which have performance guarantees and provably do not violate capacity constraints [4].

The surveys in [20,21] extensively study and categorize the earlier and more recent papers of VNE, defining a general optimization model, and proposing a taxonomy on the solutions based on their modeled features, algorithmic approaches and considered constraints. Focusing more on the technological aspects of VNE related problems, taking into account the modern developments in virtualization technologies, such as containerization, the survey in [5] presents a novel classification.

Concluding from the above state-of-the-art studies, many heuristics use a greedy approach to target easy prototypization and extension capabilities. Guarantees on the deployed service's delay are important as many emerging applications are delay sensitive. Yet, besides some exceptions [23], delay guarantees are not well-studied in the current state-of-the-art. Other solutions propose a delay-aware model [24], but the presented greedy heuristic does not rigorously address the delay guarantee, as the service delay is the optimization objective. Minimizing the service delay often disregards other important optimization goals, such as deployment cost. Authors of [25] study the trade-off between load balancing the compute infrastructure and minimizing the end-to-end service delay, but the solution lacks delay guarantees, as the SFC delay is not considered as an optimization constraint. The state-of-the-art in service placement related problems lacks heuristic solutions, where delay is taken as a constraint to meet the QoS requirements, and minimize for independent objectives.

Our recent work on service placement took the approach of including delay as an optimization constraint with configurable objective function, and implementing it in an extensible heuristic algorithm framework. In our model, delay has been considered as a path-delay, i.e., end-to-end delay, constraint in the service graph. An earlier version focuses on tackling general service graph structures in the greedy VNF placement and routing algorithm [26]. The algorithm framework has been applied to data-plane component orchestration, where the input structures are translated to the general service graph structure, and the heuristic is tuned to the domain-specific needs [27]. Addressing the network operation scenarios, where service requests continuously arrive beyond 0-day deployments, we have extended our framework with hybrid online-offline optimization capability [28]. Most recently, the algorithm has been integrated with our multi-operator 5G proof-of-concept architecture [19], and its advanced features, such as path anti-affinity, has been studied by complexity theory methodologies. Our proof-of-concept implementation is conceptually similar to the previously referred IoT slice orchestration system in [8]. In this paper, our contribution is the extension of our well-established heuristic framework with mathematically founded guarantees for the end-to-end delay constraints. Our results on end-to-end delay and infrastructure abstraction could be applied and integrated with any heuristic algorithm which uses step-by-step placement of the individual service components.

3. Base System Model and Optimization Problem

To model a wide range of the previously mentioned dynamic and heterogeneous computation resource types, we introduce four types of substrate nodes arranged in two dimensions. One dimension is the nature of the node's capacity in terms of flexibility. Substrate nodes with fixed capacity model physical servers, legacy hardware for hosting PNFs and static virtual computation capacities, etc., while elastic substrate nodes model modern pay-as-you-use cloud platforms and auto-scaling capable Virtual Infrastructure Managers (VIMs) such as a Kubernetes host.

The other dimension of infrastructure capability modeling is the level of abstraction represented by the node. A single computation node is represented by the single types, where the placed VNF is directly instantiated, e.g., a single server or a VIM managing a few VMs running on the same or proximate server racks. Finally, aggregate node types hide information of the underlying computation capabilities and show only an abstract view of multiple (possibly distant) VNF hosting possibilities. Except the aggregate elastic category, each infrastructure type is modeled in this paper and our contributions to the domain of infrastructure modeling are summarized by the considered resource types' abbreviations in Table 1. The resource types are introduced one-by-one incrementally generalizing the system model, while their challenges are studied in detail. In this section we introduce the basic system model only using the single fixed, denoted by $(s.f.)$, as this is the only type generally considered by most of the related VNE, service placement and SFC embedding literature.

Table 1. The modeled substrate node types in the dimensions of flexibility and abstraction, summarizing our contribution to the state-of-the-art (StoA) and the studied substrate node type abbreviations.

	Fixed	**Elastic**
single	$(s.f.)$ Considered in majority of SotA	$(s.e.)$ Our contribution based on M/M/c results
aggregate	$(a.f.)$ Our contribution based on BiS-BiS model	$(a.e.)$ Briefly discussed

3.1. System Model and Problem Formulation

We model the substrate network as an undirected, connected graph $S = (V_S, E_S)$, where substrate nodes V_S represent the compute resources of any type, their interconnections are the edge set $E_S \subseteq V_R \times V_R$. Notations of the problem formulation and for the rest of the paper are summarized in Table 2. The VNF repository is represented by $\mathcal{R}^{(vnfs)}$, which contains all VNFs supported by any parts of the system, used for building services. Keeping aligned with our primary motivation of modeling the characteristics of the infrastructure, we keep the service model simple. We formulate the problem only for the online variant of the service placement problem, restricted to path request graphs. We introduce a general capacity function c_S, which decides whether a set of VNFs can be placed to a substrate node due to capacity constraints. This function is incrementally extended as we introduce our proposed model for the heterogeneous substrate nodes. For now, the optimization problem is stated with only single fixed $(s.f.)$ substrate nodes, whose capacity is described by the capacity function $c_S^{(s.f.)}$, while the VNF demand mapped to these substrate nodes is described with the corresponding demand function $d_R^{(s.f.)}$. The service deployment's lifetime T_R is considered at the time of component placement decision, as it might affect the incurring costs and the substrate network's behavior. The placement cost of a VNF is specified by function p_{V_R}, taking the hosting time length and the mapping as input. Similarly, the routing cost of a request link is specified by p_{E_R}. If not stated otherwise, the costs p_{V_R} and p_{E_R} are regarded as input parameters. The node and link mapping functions, $\mathfrak{V}$ and $\mathfrak{E}$, respectively, describe the request graph embedding solution. Link throughput capacities and requirements are not modeled in this paper as we focus on node resource abstraction. Such placement problem variants can be found in our previous work [19,26–28] and the literature [20,21].

Table 2. Notations used in embedding algorithm description.

Notation	Description
$S = (V_S, E_S)$	Substrate graph structure for hosting the request R
$\mathcal{R}^{(vnfs)}$	Base set of all VNFs used to build any type of service
$R = (V_R, E_R),\ V_R \subseteq \mathcal{R}^{(vnfs)}$	Request path graph of n VNFs connecting $s_0, t \in V_S$
$\mathcal{P}(Q)$	Power set, i.e., all possible subsets, of set Q
$L_R : V_R \mapsto \mathcal{P}(V_S)$	VNF placement constraints
$P_R(e_i) \subseteq \mathcal{P}(E_S), e_i \in E_R$	Set of feasible paths for request edge e_i
$c_S : \mathcal{P}(\mathcal{R}^{(vnfs)}) \times V_S \mapsto \{0, 1\}$	General capacity function deciding hosting capabilities
$\mathcal{T}_S : V_S \mapsto \{(s.f.), (s.e.), (a.f.), (a.e.)\}$	Substrate node type of a substrate node
$c_S^{(s.f.)} : V_S \mapsto \mathbb{Z}^+$	Capacity function of single fixed substrate node types
$d_R^{(s.f.)} : \mathcal{R}^{(vnfs)} \mapsto \mathbb{Z}^+$	Demand of a VNF if mapped to a single fixed substrate node type
$T_R \in \mathbb{R}^+$	Number of time units the service is expected to run
$p_{V_R} : \mathbb{R}^+ \times \mathcal{R}^{(vnfs)} \times V_S \mapsto \mathbb{R}^+$	Mapping cost of a VNF to a substrate node for a time interval
$p_{E_R} : \mathbb{R}^+ \times E_R \times \mathcal{P}(E_S) \mapsto \mathbb{R}^+$	Mapping cost of a request edge to a substrate path for a time interval
$\mathfrak{V} : V_R \mapsto V_S$	Hosting substrate node of request node (VNF)
$\mathfrak{E} : E_R \mapsto \mathcal{P}(E_S)$	Hosting substrate path of request edge
$e_i, v_i \in E_S \times V_S = \Psi_R$	Leg, the unit orchestration step

Formulation 1 shows the service placement optimization problem, where the request graph may only be a path graph, with both ends fixed to substrate nodes from V_S as shown in (3). A demonstrative solution for Formulation 1 and some examples for the notations of Table 2 are shown in Figure 2. Constraints (1) and (2) ensure that all elements of the service are mapped. The location function L_R describes placement requirements for each VNF, which might come from operational policies, privacy restrictions, Service Access Point (SAP) matching rules or VNF functionality constraints. The set of feasible paths of a request edge $P_R(e_i)$ represents not detailed path constraints like, link capacity and routing constraints. A basic graph embedding requirement is described by (4), which ensures that the source and termination of a request link's hosting path goes between the hosts of its ends. The general capacity function c_S defines for a set of VNFs whether a substrate node has enough capacity to run all functions. Prevention of overloading any substrate node (considering all resource types for extensions) is described by (5). The general capacity function is specified in (6) for single fixed $(s.f.)$ substrate node types, where the demands are integers, simply summed and upper bounded by the integer capacity. This constraint is later extended as we introduce and study the details of other substrate node types. Finally, the objective is to minimize the overall deployment cost which is the total sum of all VNF hosting cost and all routing costs, as shown in (7).

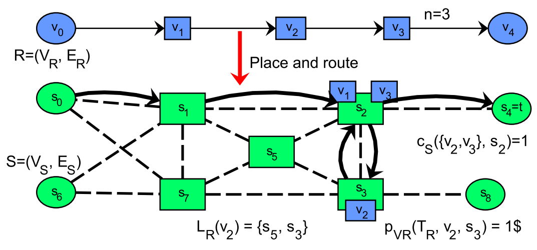

Figure 2. Demonstration of placing a path request graph, meeting the constraints of Formulation 1.

The example on Figure 2 shows concrete values for some parameters. The service chains starting and ending nodes, v_0 and v_4, respectively, have fixed locations (not subject to optimization) in the substrate graph s_0 and s_4. VNF v_2 can only be located on substrate nodes s_5 and s_3, and its cost on the latter location is 1$ for the whole service lifetime T_R.

Formulation 1 Variation of the service placement problem studied in this paper.

Input: $(V_S, E_S), (V_R, E_R), c_S, c_S^{(s.f.)}, d_R^{(s.f.)}, L_R, T_R$
Output: $\mathfrak{V}, \mathfrak{E}$

$$\forall v_i \in V_R : \exists \mathfrak{V}(v_i) \in L_R(v_i) \subseteq V_S \tag{1}$$

$$\forall e_i \in E_R : \exists \mathfrak{E}(e_i) \in P_R(e_i) \subseteq \mathcal{P}(E_S) \tag{2}$$

$$v_0, v_{n+1} \in V_R : \mathfrak{V}(v_0) = s_0, \quad \text{and} \quad \mathfrak{V}(v_{n+1}) = s_{n+1} = t \tag{3}$$

$$\forall (v_i, v_j) \in E_R : \mathfrak{V}(v_i) = \mathfrak{E}(v_i, v_j).src\_node \quad \text{and} \quad \mathfrak{V}(v_j) = \mathfrak{E}(v_i, v_j).dst\_node \tag{4}$$

$$\forall s_i \in V_S : c_S(\{\forall v_j | \mathfrak{V}(v_j) = s_i, v_j \in V_R\}, s_i) = 1 \tag{5}$$

$$\forall s_i \in V_S, r \subseteq V_R : c_S(r, s_i) = \begin{cases} 1, & \text{if } \sum_{v \in r} d_R^{(s.f.)}(v) \leq c_S^{(s.f.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (s.f.) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$\text{subject to } minimize_{\mathfrak{V}, \mathfrak{E}} \sum_{v \in V_R} \sum_{e \in E_R} p_{V_R}(T_R, v, \mathfrak{V}(v)) + p_{E_R}(T_R, e, \mathfrak{E}(e)) \tag{7}$$

3.2. Heuristic Greedy Backtrack Approach

This section briefs our greedy backtracking algorithm framework for solving service placement for various constraints in our earlier works. Here we state only the general outline of the algorithm to give sufficient background, and a previously solved service placement peculiarities are listed.

Algorithm 1 takes as input the service and substrate graph structures $R = (V_R, E_R)$ and $S = (V_S, E_S)$, and returns a complete mapping structure of VNFs and their connections $\mathfrak{V}$ and $\mathfrak{E}$, if any feasible is found. An ordered list of unit orchestration steps, i.e., *legs* (e_i, v_i), a pair of VNF and adjacent request link, is stored in list Ψ_R and generated by function ORDERLEGSFORMAPPING(V_R, E_R). In the current case the ordering is simple, as the request graph $R = (V_R, E_R)$ is a path graph between a source and a destination. In more complicated applications this function returns the legs in an order so that their source is always placed by the MAPONENF(e_i, v_i) function, before their own placement decisions. We note that our results on the substrate graph model are general and independent of the service graph structure. Previously published and applied versions of the greedy backtrack approach are capable of handling general service structures [19,26–28]. The MAPONENF(e_i, v_i) function generates hosting candidates and filters them according to the constraints, as shown in Formulation 1. In case such a greedy step is not successful, the functions GETBACKTRACKOPTION(e_i, v_i) and UNDOGREEDYMAPPING(e_i, v_i) are responsible to explore other greedy mapping directions and to maintain the resource reservations, constraint states and mapping structures. When a complete mapping structure solving Formulation 1 is found, Algorithm 1 returns the solution. Our earlier work also studies tackling the runtime complexity of such a greedy mapping approach for the NP-hard service placement problem [19,27], these examinations are out of the scope of this paper.

In our earlier work we used this greedy heuristic service placement algorithm framework in various different settings, including general service topology structure [26], data plane resource orchestration [27], hybrid online-offline operational setting [28], and integrated it into a 5G network architecture supporting advanced resource (anti)-affinity constraints [19]. Our greedy heuristic framework has served as basis for several patents submitted by Ericsson, patenting VNF decomposition option selection, elastic substrate capacity management and VNF interconnection constraints (Patents pending).

The versatility of the greedy backtracking approach allows it to be applied in various service graph embedding scenarios, including 5G-enabled IoT prototype architectures independent of our recent work [8]. Our algorithm framework can be easily extended, ideal for early deployment, prototype architecture integration and continuous development.

In this paper, we propose theoretical foundations to the greedy approach by providing guarantee on the embedding result's end-to-end delay requirement. Subsequently, we present our mathematical analysis on node capacity allocation in a heterogeneous distributed computing environment by refining the substrate graph model.

Algorithm 1 Overview of the Greedy Backtrack Metaheuristic for VNE

```
    Returns a set of complete mapping structures of service graph $S$ to resource graph $R$.
 1: procedure MAP($S, R$) $\rightarrow \mathfrak{V}, \mathfrak{E}$
 2:     $\Psi_R \leftarrow$ ORDERLEGSFORMAPPING($V_R, E_R$)          ▷ Lists all VNFs and links to be mapped
 3:     while $\exists e_i, v_i \in \Psi_R$ where $\nexists \mathfrak{V}(v_i)$ or $\nexists \mathfrak{E}(e_i)$ do
 4:         while MAPONENF($e_i, v_i$) not successful do
 5:             $e_{i'}, v_{i'} \leftarrow$ GETBACKTRACKOPTION($e_i, v_i$)
 6:             UNDOGREEDYMAPPING($e_{i'}, v_{i'}$)
 7:             $e_i, v_i \leftarrow e_{i'}, v_{i'}$                        ▷ Try remapping another leg
 8:         end while
 9:     end while
10:     return $\mathfrak{V}, \mathfrak{E}$
11: end procedure
```

4. Result on Path Delay Constraint

This section extends our system and optimization model with latency values, formalizes path delay constraint and we present our result on how to design the step-by-step delay heuristics to approximate end-to-end delay requirements. The notations related to the delay model are shown in Table 3.

Table 3. Notations for the end-to-end latency results.

Notation	Description
$d_{s_i s_j},\ s_i, s_j \in V_S$	Delay distance of two substrate nodes
$d^{(fp)}_{s_i s_j},\ s_i, s_j \in V_S$	Delay of a feasible path between two substrate nodes.
$\mathfrak{V}(v_0) = s_0, \mathfrak{V}(v_{n+1}) = s_{n+1} = t$	Source and target of the path request graph.
$D^{(req)}, D^{(used)}_{n+1}$	Required and used path delay on VNF path $V_R = \{v_0, \dots v_n, v_{n+1}\}$.
$d^{(\mathfrak{E})}_{v_j \rightsquigarrow v_{j+1}} = d^{(fp)}_{\mathfrak{V}(v_j)\mathfrak{V}(v_{j+1})}$	Delay of the hosting substrate path between $\mathfrak{V}(v_j)$ and $\mathfrak{V}(v_{j+1})$
$D^{(\mathfrak{V}(v_i))} = D^{(req)} - \sum_{j=0}^{i-1} d^{(\mathfrak{E})}_{v_j \rightsquigarrow v_{j+1}}$	Remaining delay budget while all VNFs until v_i are mapped.

We annotate each substrate graph connection by communication delays, and calculate the distances of each substrate node pairs $\forall s_i, s_j \in V_S$, denoted by delay distance $d_{s_i s_j}$. In our general greedy framework of Algorithm 1, the MAPONENF(e_i, v_i) function can allocate any path $\mathfrak{E}(e_i)$ for hosting the request edge $e_i \in E_R$, which is aligned with path constraints (see $P_R(e_i)$). The delay of such feasible path is denoted by $d^{(fp)}_{s_i s_j}$. This substrate graph path delay for hosting a request graph edge is alternatively denoted by $d^{(\mathfrak{E})}_{v_j \rightsquigarrow v_{j+1}}$, to eliminate the indirection of the node mapping function $\mathfrak{V}$ from the notation. Having the VNFs indexed in their order on the path request R as

$V_R = \{v_0, \dots v_n, v_{n+1}\}, E_R = \{(v_0, v_1), \dots (v_n, v_{n+1})\}$, we extend the original problem definition of Formulation 1 by a path delay requirement.

$$D^{(used)}_{n+1} = \sum_{(v_i, v_{i+1}) \in E_R} d^{(\mathfrak{E})}_{v_j \rightsquigarrow v_{j+1}} \leq D^{(req)} \tag{8}$$

where $D^{(req)}$ denotes the maximal allowed path delay of the used route from the beginning to the end of the service graph path.

Assume that the function ORDERLEGSFORMAPPING(V_R, E_R) returns the edges of the path request graph in order, starting from the fix substrate node $s_0 = \mathfrak{V}(v_0)$, where the chain starts from. The algorithm updates the remaining delay budget $D^{(\mathfrak{V}v_i)}$ by subtracting the used path delay $d^{(fp)}_{\mathfrak{V}(v_{i-1})\mathfrak{V}(v_i)}$ after the greedy mapping of a leg leading to VNF v_i. The termination of the request path graph R is in substrate node $\mathfrak{V}(v_{n+1}) = s_{n+1}$, which is alternatively referred to as t. We state our result on delay guarantee in the following theorem.

Theorem 1 (End-to-end delay constraint guarantee)**.** *A greedy heuristic maps a VNF in each step $\forall v_i \in \{v_1, \dots v_n\}$, where all VNFs until index i are greedily placed, i.e., $\exists \mathfrak{V}(v_i)$, and the hosting substrate node and path of leg (e_{i+1}, v_{i+1}) are being selected.*

During the greedy mapping of a service graph, if for choosing the host of all VNFs $v_{i+1} \in \{v_1, \dots v_n\}$,

- *used delay is a fraction of the path delay requirement: $d^{(\mathfrak{E})}_{v_j \rightsquigarrow v_{j+1}} \leq \alpha D^{(req)}, \quad \frac{1+\epsilon}{n+1} < \alpha \leq 1$,*
- *the used substrate node's distance is bounded: $d_{\mathfrak{V}(v_i)\mathfrak{V}(v_{i+1})} + d_{\mathfrak{V}(v_{i+1})t} > B^{(dist)}$, and $d_{\mathfrak{V}(v_{i+1})t} \leq d_{s_0 t}$*
- *and the remaining delay requirement is bounded: $D^{(\mathfrak{V}(v_i))} > B^{(req)}$,*

then the used total path delay $(1+\epsilon)$ approximates the path delay requirement:

$$D^{(used)}_{n+1} \leq (1+\epsilon) D^{(req)}, \qquad \epsilon \in \mathbb{R}^+,$$

where

$$0 < B^{(dist)} < \frac{n+\alpha}{n+1} D^{(req)}, \tag{9}$$

$$0 < B^{(req)} < \alpha D^{(req)} + d_{s_0 t}, \tag{10}$$

$$B^{(dist)} B^{(req)} \geq \frac{n+\alpha-\epsilon}{n+1} (\alpha D^{(req)} + d_{s_0 t}) D^{(req)}. \tag{11}$$

The proof of the Theorem 1 is presented in Appendix A.

Theorem 1 helps designing a heuristic algorithm's step-by-step bounds on the latency values to guarantee the $(1+\epsilon)$-approximation of the end-to-end path delay constraint. It is important to note that the algorithm does not need to minimize for delay to comply with the constraint, i.e., shortest path calculation for each request edge might be done for the cost metric, while keeping the theorem's bounds on the delay. The corresponding subproblem is the restricted shortest path problem, for which efficient algorithms have been proposed [29].

Furthermore, the presented result does not utilize the previously well-studied backtracking capability of the proposed embedding framework. If substrate elements with the desired bounds can not be found, or the $(1+\epsilon)$-violation of the delay constraint needs to be mitigated, undoing some earlier greedy placement steps may solve the issue. Relation between the $(1+\epsilon)$ approximation ratio and the required search space for the backtracking might be studied to design a fully polynomial-time approximation scheme (FPTAS) for the problem. A numerical example for applying Theorem 1 is shown in Section 7.

Having demonstrated how advanced constraints, such as end-to-end delay, can be supported by the greedy backtrack framework, now we turn to refining the substrate graph model which provides powerful benefits for the embedding algorithm.

5. Result on Resource Aggregation

5.1. Revisit the BiS-BiS Model

Network abstractions of various types have proved useful in the past, such as network abstraction map [12], hose model for VPN design [11] and more recently the Big Switch abstraction in SDN [15]. The Big Switch model describes network segments with their end-to-end transport characteristics. As computation and networking are increasingly interleaving thanks to softwarization and virtualization, the Big Switch approach was extended to describe both by the Big Switch-Big Software (BiS-BiS) abstraction [17,18]. Our work refines the BiS-BiS abstraction and provides theoretic foundation to it.

To describe an abstract view of distributed computation sites and to supply valuable information for latency sensitive applications, we propose adding internal aggregate nodes to the BiS-BiS model which describe transport and/or computation resources. Figure 3 shows an abstract view of the lower level infrastructure view of Figure 1. In this example, the CSP chooses to show its network aggregated into two BiS-BiS domains, describing the west and east segments of its infrastructure by CSP BiSBiS1 and CSP BiSBiS2, respectively. The CSP's three computing locations, Server2, Server3 and Server4 of Figure 1 are aggregated into the CSP Aggr. node of Figure 3. The other non-aggregated nodes, Server1, DataCenter1 and Cluster1 of Figure 1 are represented by CSP Single, DC and OTT Priv nodes, respectively. To simplify the networking structure and to hide some lower level topological information, forwarding nodes of the infrastructure are also aggregated. For instance, the two abstract forwarding nodes of CSP BiSBiS2 in Figure 3 represent the eastern side of the forwarding topology in Figure 1. The edges connecting the internal nodes of a BiS-BiS are annotated by delay values, describing the transport characteristics. Our embedding algorithm proposed in Section 3 is able to operate on a substrate graph, which has such BiS-BiS domains, and is able to map service graphs with end-to-end delay constraints as described in Section 4. The extensions required for the capacity constraints (6) to model the aggregate capacity are presented in this section.

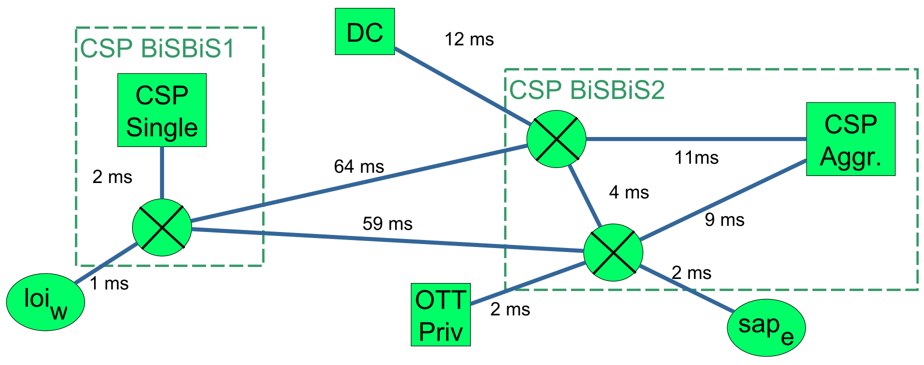

Figure 3. Demonstration of modeling an abstract view of the heterogeneous computation and network infrastructure (rented from the CSP and CP) of Figure 1. This figure shows the topological information which is used by the OTT SP's orchestration logic to deploy services across the abstract multi-domain view.

We assume that defining the set of substrate nodes to be aggregated is out-of-scope of the underlying optimization problem, as the clustering might be more influenced by the CSP's operational policies than rigorous mathematical measures. An extensive survey on clustering algorithms in sensor networks can be found in the literature [30], while general graph clustering is studied in [31].

As it was shown in previous experiments on abstract infrastructure views [19], communicating the bottlenecks is in the interest of both the OTT SP and the CSP to avoid overbooking and under-utilization. To facilitate the effective communication of the bottlenecks, the aggregation must hold an essential property shown in Remark 1.

Remark 1 (Essential property of substrate abstraction)**.** *If a set of VNFs composing the service graph* R *seems to be hostable on an aggregate substrate node, then the VNF set must be able to be mapped to the underlying hidden network segment.*

Given the clustering of the underlying substrate nodes which are to be aggregated, the central problem in creating the BiS-BiS view is the aggregation of the hidden nodes' capacities into a single abstract node's capacity. The aggregating BiS-BiS view must comply with Remark 1 and maximize the shown capabilities of the hidden network segment. In the next section we formulate this central problem as the resource contraction problem, and propose an effective solution to it.

5.2. Resource Contraction Problem

To formulate the resource abstraction problem as an optimization we first restate and generalize the capacity allocation subproblem of Formulation 1, characterized by (1), (5) and (6).

First, the capacity allocation problem should be independent of the request graph instance $R = (V_R, E_R)$, as the aggregation view should be applicable to any service supported by the system. So we introduce a multiset of the base VNF set $(\mathcal{R}^{(vnfs)}, m_R)$ including not only any set of VNFs but also their repetitions m_R. VNFs in an IoT service might appear multiple times, e.g., data processing and noise filtering might need to be done in multiple distributed locations, close to the sensors or end-users. Notations corresponding to this chapter are shown in Table 4. The set of substrate nodes to be aggregated is denoted by $V'_S \subseteq V_S$. Formulation 2 shows the capacity allocation problem generalized to multisets of the possible VNFs and arbitrary subsets of the substrate graph. The equations are formalized for a listing of the request multiset $(\mathcal{R}^{(vnfs)}, m_R)$, where elements with higher multiplicity are repeated accordingly. All repetitions of all VNF v_i must be placed somewhere, as expressed by (13). The problem only checks whether an allocation is possible, solving a subproblem of Formulation 1 with (13) and (14) relating to (1) and (5), (6) respectively.

Table 4. Notations for modeling the Big Switch-Big Software (BiS-BiS) nodes.

Notation	Description
$(Q, m_Q),\ m_Q : Q \mapsto \mathbb{N} \cup \{0\}$	Multiset of set Q, with multiplicity function m_Q
$size((Q, m_Q)) = \sum_{q \in Q} m_Q(q)$	Size of a multiset, i.e., including item multiplicity
$V'_S \subseteq V_S$	Subset of the substrate nodes for allocation/aggregation
$\texttt{CAP}(V_S', (Q, m_Q), c_S^{(s.f.)}, d_R^{(s.f.)})$	Shorthand of solving the problem of Formulation 2
$cap\_sum(V'_S) = \sum_{s_i \in V'_S} c_S^{(s.f.)}(s_i)$	Sum of capacities of a substrate node set V_S'
$dem\_sum((V'_R, m_{R'})) = \sum_{v_i \in V'_R} d_R^{(s.f.)}(v_i) m_{R'}(v_i)$	Sum of demands of VNF multiset $(V_R', m_{R'})$
$c_S^{(a.f.)} : V_S \mapsto \mathbb{Z}^+$	Capacity function of aggregate fixed type substrate nodes

Formulation 2 Capacity Allocation Problem (CAP)

Input: $V'_S, (\mathcal{R}^{(vnfs)}, m_R), c_S^{(s.f.)}, d_R^{(s.f.)}$
Output: True if $\exists\, \mathfrak{V}$, False otherwise

$$N := size((\mathcal{R}^{(vnfs)}, m_R)) \tag{12}$$

$$\forall v_i \in \{v_1, v_2, \dots v_N\} : \exists \mathfrak{V}(v_i) \in V'_S \tag{13}$$

$$\forall s_i \in V'_S : \sum_{v_j | \mathfrak{V}(v_j) = s_i, v_j \in \{v_1, \dots v_N\}} d_R^{(s.f.)}(v_j) \leq c_S^{(s.f.)}(s_i) \tag{14}$$

The resource contraction problem shown in Formulation 3 builds on the capacity allocation problem. It takes as input a subset of the substrate nodes $V'_S \subseteq V_S$ to be aggregated, and returns an aggregate fixed substrate node $\bar{s}$ with resource type $\mathcal{T}_S(\bar{s}) = (a.f.)$ and aggregate capacity $c_S^{(a.f.)}(\bar{s}) = A$. Such an example is shown in Figure 3, where the CSP decides to aggregate its single fixed type servers in Figure 1 into a single aggregate substrate node. This is contained in the BiS-BiS created for the western part of the CSP's infrastructure and is connected to the other parts of the substrate graph through aggregate forwarding nodes shown in Figure 3. The essential property of substrate abstraction of Remark 1 is formalized in (16). For all VNF sets with any multiplicity, if it seems to be placeable on the aggregate node $\bar{s}$ using the Capacity Allocation Problem (CAP), then it must be possible to allocate it to the hidden nodes. The capacity of the aggregate node shall be maximized to show as accurate view as possible, which is formalized by the objective in (17). The capacity of the aggregate node should never exceed the sum of the hidden nodes' capacities, formalized in (15). Note that for evaluating the condition of (16), the Equation (14) simplifies to $dem\_sum((V_R{}', m_{R'})) \leq c_S^{(s.f.)}(\bar{s}) = c_S^{(a.f.)}(\bar{s}) = A$.

Formulation 3 Resource contraction problem (RCP)

Input: $V'_S, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)}$
Output: $A, \bar{s}, c_S^{(a.f.)} : \{\bar{s}\} \mapsto \{A\}, \mathcal{T}_S(\bar{s}) = (a.f.)$

$$A \in \mathbb{Z}^+ : A \leq \sum_{s_i \in V'_S} c_S^{(s.f.)}(s_i) \tag{15}$$

$$\begin{cases} \forall V'_R & \subseteq \mathcal{R}^{(vnfs)}, \forall (V'_R, m_{R'}) : \\ \text{If} & \mathtt{CAP}(\{\bar{s}\}, (V'_R, m_{R'}), c_S^{(s.f.)}(\bar{s}) = A, d_R^{(s.f.)}) \text{ is True then} \\ & \mathtt{CAP}(V'_S, (V'_R, m_{R'}), c_S^{(s.f.)}, d_R^{(s.f.)}) \text{ is True} \end{cases} \tag{16}$$

$$\text{subject to } maximize_{A \in \mathbb{Z}^+} A \tag{17}$$

To this end, if the RCP is solved for a set of single fixed type $(s.f.)$ substrate nodes $V'_S \subseteq V_S$ with capacity function $c_S^{(s.f.)}$ and its accompanying demand function $d_R^{(s.f.)}$, then their contracted view is the aggregate fixed type $(a.f.)$ substrate node $\bar{s}$ with capacity A. Note that it is not necessary to define a different demand function for placing VNFs on aggregate fixed nodes, i.e., $d_R^{(a.f.)} = d_R^{(s.f.)}$. Instead of showing the whole set of substrate nodes V'_S, the CSP can choose to publish the aggregate node, which integrates to the resource view of the OTT SPs.

Extending the substrate graph model with the aggregation node $\bar{s}$ specified by the resource contraction problem, we extend Formulation 1 to include aggregate fixed nodes in (6) as shown in (18):

$$\forall s_i \in V_S, r \subseteq V_R : c_S(r, s_i) = \begin{cases} 1, & \text{if } \sum_{v \in r} d_R^{(s.f.)}(v) \le c_S^{(s.f.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (s.f.) \\ 1, & \text{if } \sum_{v \in r} d_R^{(s.f.)}(v) \le c_S^{(a.f.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (a.f.) \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

This describes a heterogeneous view of the infrastructure, while hiding some topology information as desired. The view might be periodically updated as the capacities saturate, to keep the shared information up-to-date and useful, according to the agreement among the providers.

Similar to many assignment problems, the CAP, and thus the RCP as well are NP-hard. The CAP is a subproblem of the notoriously hard VNE problem, and proven to be NP-hard by reducing it to the famous 3-SAT problem [3]. In the following we present an approximation algorithm for solving the RCP, and we analyze its performance.

5.3. Approximating Resource Contraction

We propose a solution to efficiently approximate the NP-hard resource contraction problem (RCP) presented in Formulation 3.

The pseudo-code of APRXRCP is shown in Algorithm 2. The VNF $v_{\max}$ with largest demand $d_{\max}$ is selected from the input base VNF set. In lines 6 and 8 the big and small capacity substrate nodes are defined, based on being or not being able to fit $d_{\max}$, defining a 2-partition of the to-be-aggregated nodes V'_S. Note that operating on integer demands and capacities is important to achieve the approximation result. A component of the final aggregation capacity A_0 is defined by accounting for at least $d_{\max}$ space in each big node $V_S'^{(big)}$ in line 7. This way, the maximum amount of repetition of the biggest VNF $v_{\max}$ is handled in the input multiset $(\mathcal{R}^{(vnfs)}, m_R)$ of CAP while solving the RCP. In addition, this approach ensures that the CAP on the hidden substrate nodes is possible, as a lower limit on the sum of such a capacity allocation is calculated. The function APRXRCP runs recursively by accounting for the highest amount of reservable capacities for the second largest VNF, by removing $v_{\max}$ from the base VNF set $\mathcal{R}^{(vnfs)}$ and executing the whole function only for the small substrate nodes $V_S'^{(small)}$ in line 10. As the recursively returned aggregation component A_1 cannot be higher than $d_{\max}$ (otherwise the essential property of Remark 1 might be violated), a result is produced. A termination criterion for the recursion is an empty input VNF set, which results in summing the capacities of the input substrate nodes (recursively remaining nodes with small capacities) in line 3.

In each recursive call, at least one VNF, the current $v_{\max}$, is taken out of the input VNF set, so the recursion is at most $|\mathcal{R}^{(vnfs)}|$ deep. The algorithm performs polynomial number of steps in the input sizes in each recursive call, so the recursion always terminates and its complexity is $\mathcal{O}(|\mathcal{R}^{(vnfs)}|^2 |V'_S|)$. We state the performance guarantee of APRXRCP presented in Algorithm 2 in relation to the optimal solution of RCP.

Theorem 2 ($\frac{1}{2}$-approximation of the Resource Contraction Problem)**.** *Given any subset of the substrate graph nodes $V'_S \subseteq V_S$ and base VNF set $\mathcal{R}^{(vnfs)}$ with capacity function $c_S^{(s.f.)}$ and VNF demand function $d_R^{(s.f.)}$, Algorithm 2 $\frac{1}{2}$-approximates the optimal solution of the Resource Contraction Problem of Formulation 3:*

$$\frac{1}{2} OPT\{\texttt{RCP}(V'_S, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)})\} \le \text{APRXRCP}(V'_S, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)}) \tag{19}$$

The proof of Theorem 2 is presented in Appendix B. A numeric example for Formulation 3 solved by APRXRCP is presented in Section 7.

Supported by Theorem 2, we can efficiently calculate the capacity of an aggregate fixed $(a.f.)$ type substrate node, hiding a set of single fixed $(s.f.)$ type substrate nodes. This enables a CSP to provide a heterogeneous infrastructure view to its tenant OTT SPs, hiding sensitive parts of the topology, while being able to show exact views where it is necessary, as it is demonstrated in Figure 3. The CSP can use APRXRCP to $\frac{1}{2}$-approximate the optimal aggregate capacity to be shown to its tenants' resource orchestrators. The presented greedy heuristic framework Algorithm 1 in Section 3.2 can be easily extended to respect the multiple types of substrate nodes, because only the capacity calculation function of Formulation 1 needs to be modified. The aggregation scheme also contributes to the scalability of the embedding algorithm executed by the OTT SP, as it decreases the number of substrate nodes in its infrastructure view. As shown in (18), the capacity interpretation is extended for $(a.f.)$ type nodes and its implementation can be done in the MAPONENF(e_i, v_i) function.

Algorithm 2 Resource contraction algorithm solving Formulation 3

```
    Input: $V'_S, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)}$
    Output: $A$
 1: procedure APRXRCP($V'_S, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)}$) $\to A$
 2:     if $\mathcal{R}^{(vnfs)} = \emptyset$ then
 3:         return $cap\_sum(V_S')$
 4:     else
 5:         $v_{\max}, d_{\max} \leftarrow \max_{d_R^{(s.f.)}} \{\mathcal{R}^{(vnfs)}\}$        ▷ Get the VNF with biggest demand
 6:         $V_S'^{(big)} \leftarrow \{s_i \mid s_i \in V'_S,\ d_{\max} \leq c_S^{(s.f.)}(s_i)\}$
 7:         $A_0 \leftarrow \sum_{s_i \in V_S'^{(big)}} \max\left\{\left\lfloor \frac{c_S^{(s.f.)}(s_i)}{2} \right\rfloor + 1,\ c_S^{(s.f.)}(s_i) - d_{\max} + 1\right\}$
 8:         $V_S'^{(small)} \leftarrow \{s_i \mid s_i \in V'_S, d_{\max} > c_S^{(s.f.)}(s_i)\}$
 9:         if $V_S'^{(small)} \neq \emptyset$ then
10:             $A_1 \leftarrow$ APRXRCP($V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\}, c_S^{(s.f.)}, d_R^{(s.f.)}$)
11:             return $A_0 + \min\left\{d_{\max} - 1,\ A_1\right\}$
12:         else
13:             return $A_0$
14:         end if
15:     end if
16: end procedure
```

6. Results on Elastic Resources

6.1. Modeling Resource Elasticity

To capture important characteristics of state-of-the-art computation infrastructure, we model elastic capacity and on-demand billing. Kubernetes is widely used in the industry today, thanks to its active open source community, enabling the rapid development of practical IoT deployments. As it was demonstrated earlier, Kubernetes-based virtualization architectures are a good candidate to meet the extreme requirements of IoT services [8]. We take the widely used Horizontal Pod Autoscaler (HPA) of the Kubernetes container management system [6] as a representative example to model resource elasticity. We propose a mathematical model of Kubernetes HPA to extend our detailed model of the heterogeneous compute and network infrastructure.

Using Kubernetes terminology, we refer to a single instance of a containerized VNF as pod, which is managed by HPA and replicated or terminated in response to changing workload. By default, HPA measures the average of all instantiated pods' utilization of their allocated CPU capacity, and compares it to a target utilization [7].

Definition 1 lists the HPA parameters and formalizes its operation, which is the base for describing the behavior of a single elastic substrate node $s_i \in V_S$.

Definition 1 (Kubernetes HPA parameters and operation)**.** *The following parameters are configured by the operator of HPA [7]:*

1. *c_{min} and c_{max} are the minimal and maximal allowed pod count,*
2. *$\hat{u} \in [0,1]$ is the targeted scaling metric average of all running pods,*
3. *$t^{(sca)} \in \mathbb{R}^+$ time length of a scaling interval, when HPA evaluates metrics for scaling decisions,*
4. *$\sigma \in [0,1]$ is the scaling tolerance,*
5. *$N^{(sca)} := \left\lfloor \frac{T_R}{t^{(sca)}} \right\rfloor, T_R > t^{(sca)}$ is the number of scaling intervals during the lifetime T_R of the service R.*

$u_j \in [0,1]$ is the measured scaling metric average of all running pods in scaling interval $j \in \{1,2,\dots N^{(sca)}\}$. A scaling decision is made in scaling interval j, when

$$\left| \frac{u_j}{\hat{u}} - 1 \right| > \sigma, \tag{20}$$

and the number of pods in interval $j+1$ recursively yields:

$$c^*_{j+1} = \left\lceil c_j \frac{u_j}{\hat{u}} \right\rceil. \tag{21}$$

After applying the limits, pods are terminated or instantiated to meet the count

$$c_{j+1} = \textsc{applyPodLimits}(c^*_{j+1}) = \begin{cases} c^*_{j+1} & \text{if } c_{min} \le c^*_{j+1} \le c_{max} \\ c_{min} & \text{if } c^*_{j+1} < c_{min} \\ c_{max} & \text{if } c^*_{j+1} > c_{max}. \end{cases} \tag{22}$$

During a given scaling interval, the number of pods (i.e., VNF replicates) processing the incoming task of a VNF v_i is fixed. Elementary tasks of a VNF model are for example sensor data processing tasks, video processing requests, HTTP requests for a webserver VNF, flow classification requests for a firewall VNF or incoming images for an image processing VNF, etc. We extend our VNF model by assigning incoming elementary task rate λ_i and processing rate μ_i to VNF v_i. Notations for the service and substrate model extensions of this section are gathered in Table 5.

Table 5. Notations for modeling the elasticity of substrate nodes.

Notation	**Description**
$\lambda_i \; \forall v_i \in V_R$	Incoming elementary task rate of VNF v_i
$\mu_i \; \forall v_i \in V_R$	Processing rate of elementary tasks of VNF v_i running in a single instance
$s_i :: [x, y],\; s_i \in V_S$	Local parameters x and y for substrate node s_i, e.g., parameters of Definition 1
$c_j, c^*_j, c_{min}, c_{max}, c^{(curr)}$	Pod numbers in various situations
$u_j, \hat{u}, u^{(curr)}, \tilde{u}$	Scaling metric in various situations
$c^{(s.e.)}_S : V_S \mapsto \mathbb{Z}^+$	Capacity function of single elastic substrate node type.
$d^{(s.e.)}_R : \mathbb{R}^+ \times V_R \times V_S \mapsto \mathbb{N} \cup \{0\}$	Demand function of a VNF to-be-placed on $(s.e.)$ type substrate node

As the default configuration of HPA uses CPU resources as scaling metric, we primarily assume the modeling characteristics based on the nature CPU utilization of computation intensive tasks. We propose to model the pod utilization states using a classic M/M/c queue, where "c" number of pods are processing the VNF's incoming tasks in parallel. The corresponding assumptions are collected in Proposition 1.

Proposition 1 (Model scaling interval as M/M/c queue)**.** *A scaling interval of the Kubernetes HPA $j \in \{1, 2, \dots N^{(sca)}\}$ for a single VNF $v_i \in V_R$ can be described by a classic M/M/c model supposing the following assumptions:*

1. *the pods are instantiated and terminated instantly,*
2. *the arrival process of the VNF's task is memoryless with constant rate $\lambda = \lambda_i$ during the scaling interval,*
3. *each pod has identical characteristics, including the VNF's task processing with an identical, memoryless process with constant rate $\mu = \mu_i$ during the scaling interval,*
4. *the task queue of VNF v_i has infinite length and the task-to-pod assignment takes negligible time, assuming computation intensive tasks and CPU utilization scaling metrics, the pods have two distinct utilization states: busy and idle, with 100% and 0% scaling metric utilization, respectively,*
5. *scaling metric $\mathfrak{u}_j$ is the average utilization of all running pods.*

Note that the HPA utilization metric can be easily configured and thus the assumptions of Proposition 1 would still hold for the general scaling metric. For instance, if the application is I/O intensive, and HPA is configured to scale based on I/O resource usage, Proposition 1 remains realistic.

We state our analytical results on the M/M/c queue model in Theorem 3.

Theorem 3 (Analysis of M/M/c queue)**.** *If inequality $c\mu \leq \lambda$ holds for and M/M/c queue with arrival rate λ and "c" instances of servers with processing rate μ, then the expected values $\mathbb{E}$ of the following real random variables exist and can be calculated as follows:*

- *Total busy time Θ_c: measures the length of time until the first idle moment of an M/M/c system starting from c busy servers.*

$$\mathbb{E}[\Theta_c] = -\sum_{k=1}^{c} \frac{d}{ds} \eta_k(0)$$

- *Weighted total busy time Ω_c: measures the area under the time–busy server count chart for an M/M/c system starting from c busy servers until the first idle moment.*

$$\mathbb{E}[\Omega_c] = -\sum_{k=1}^{c} \frac{d}{ds} \beta_k(0)$$

where $\eta_k(s)$ and $\beta_k(s)$ are the Laplace–Stieltjes Transform of the random variables composing the total busy time Θ_c and weighted total busy time Ω_c respectively. The exact definitions of the random variables and their calculations are shown in Appendix C.

The proof of Theorem 3 is detailed in Appendix C. We propose to use the expected values $\mathbb{E}[\Theta_c]$ and $\mathbb{E}[\Omega_c]$ to estimate the scaling metric of HPA.

Proposition 2 (Usage of Theorem 3 to model HPA)**.** *The measured scaling metric $\mathfrak{u}_j$ of Kubernetes HPA in any scaling interval $j \in \{1, 2, \dots N^{(sca)}\}$ for a single VNF $v_i \in V_R$ can be estimated:*

$$\mathfrak{u}_j \approx \tilde{\mathfrak{u}} = \frac{\mathbb{E}[\Omega_c]}{c\mathbb{E}[\Theta_c]}$$

calculated for an M/M/c queue with server count, arrival and departure rates $c = c_j$, $\lambda = \lambda_i$ and $\mu = \mu_i$, respectively.

Argumentation. *The momentary utilization of an M/M/c system in any given time–busy server count trajectory (i.e., a concrete realization of the random process) is calculated by the number of busy servers* *(pods) divided by the number of total servers "c". Thus, the average utilization in an interval t can be calculated by the area below the time–busy pod count chart until t, divided by the maximal possible (rectangular) area ct. As Ω_c measures the area under the trajectory chart and Θ_c measures the time until the idle state, their fraction gives the average utilization. Applying the assumptions of Proposition 1 this metric equals to the scaling metric which would be measured by HPA on the given pod count trajectory. This estimation neglects the dependence of the two random variables and assumes that the probability $\mathbb{P}(t > \Theta_c)$ is negligible, which are realistic assumptions in a practical setting. Theorem 3 states the expected values $\mathbb{E}[\Theta_c]$ and $\mathbb{E}[\Omega_c]$, which can be used to estimate the measured scaling metric $u_j \approx \tilde{u}$.*

6.2. Describing Elastic Resource Behavior

As described in Definition 1, the only non-deterministic part of HPA is the scaling metric measurement u_j in scaling interval j. Having the estimation $u_j \approx \tilde{u}$ as shown in Proposition 2, we can model an elastic node's behavior in response to various loads.

Algorithm 3 determines a single elastic substrate node s_i's behavior (https://github.com/nadamdb/k8s-hpa-modeling) by the trajectory of the varying pod numbers working on a single to-be-placed VNF's task during the whole service lifetime T_R. Procedure MODELHPABEHAVIOR takes as input a VNF's intensity characteristics, and a possible elastic host node's HPA settings. It iteratively uses the subroutine NEXTHPAPODCOUNT to determine the nature of the scaling decision in each scaling interval during the lifetime of the service. The pod count trajectory function $f^{(pod)}$ is continuous in its first argument, which tells for each time instance the predicted number of running pods which would process the VNF v_i's tasks on substrate location s_i. If in any scaling interval the pod count is restricted by the maximal number of pods c_{max}, the trajectory is invalidated by terminating the algorithm according to the condition in line 7. As shown in line 17, the algorithm has to handle scaling intervals when the M/M/c model's stability criterion, i.e., a condition of Theorem 3 is not met. A cumulative overload parameter $\rho^{(overload)}$ is used between iterations, which compensates the estimated utilization metric by adding the volume of the overload. Line 21 gets the scaling metric estimation based on Proposition 2, and the overload parameter is decreased by the compensated amount. The scaling decision and the number of running pods are determined according to the HPA operation shown in Definition 1 and formalized in Algorithm 3 starting from line 24.

Note that Proposition 1 only requires the arrival and processing rates to be constant during the scaling interval. Algorithm 3 is able to handle different rates in each scaling interval, enabling it to follow dynamic changes in the request arrival or changes in the computation circumstances. The presented algorithm has been tuned to real measurements taken on a Kubernetes HPA to properly model the overloaded states, when Theorem 3 cannot be used. The overload parameter $\rho^{(overload)}$ aligns the model's prediction with experiments. A validative comparison between the estimated pod count trajectory of Algorithm 3 and a measurement on the real pod count trajectory of a Kubernetes cluster managed by HPA is discussed in Section 7.

Algorithm 3 Pod count trajectory modeling of Kubernetes HPA

```
    Input: $v_i, s_j$, parameters for $s_j$ defined in Definition 1
    Output: $f^{(pod)} : [0, T_R] \times V_R \times V_S \mapsto \mathbb{N} \cup \{0\}$
 1: procedure MODELHPABEHAVIOR($v_i, \lambda_i, \mu_i, s_j :: [\mathsf{c}_0, \mathsf{c}_{min}, \mathsf{c}_{max}, \hat{u}, t^{(sca)}, \sigma, N^{(sca)}]$)
 2:     $\mathsf{c}^{(curr)} \leftarrow \mathsf{c}_0$
 3:     $\forall t \in [0, t^{(sca)}] : f^{(pod)}(t, v_i, s_i) = \mathsf{c}_0$      ▷ Initialize output HPA trajectory function
 4:     $\rho^{(overload)} \leftarrow 0$      ▷ Cumulative overload parameter
 5:     for $k \in \{1, 2, \dots N^{(sca)}\}$ do
 6:         $\mathsf{c}_k^*, \rho^{(overload)} \leftarrow$ NEXTHPAPODCOUNT($\mathsf{c}^{(curr)}, \lambda_i, \mu_i, \hat{u}, \sigma, \rho^{(overload)}$)
 7:         if $\mathsf{c}_k^* > \mathsf{c}_{max}$ then
 8:             return $\perp$      ▷ Invalidate trajectory function $f^{(pod)}$
 9:         else
10:             $\mathsf{c}^{(curr)} \leftarrow$ APPLYPODLIMITS($\mathsf{c}_k^*, \mathsf{c}_{min}, \mathsf{c}_{max}$)      ▷ Use HPA pod limits in (22)
11:             $\forall t \in (kt^{(sca)}, (k+1)t^{(sca)}] : f^{(pod)}(t, v_i, s_i) = \mathsf{c}^{(curr)}$
12:         end if
13:     end for
14:     return $f^{(pod)}$
15: end procedure
16: procedure NEXTHPAPODCOUNT($\mathsf{c}^{(curr)}, \lambda_i, \mu_i, \hat{u}, \sigma, \rho^{(overload)}$)
17:     if $\lambda_i > \mathsf{c}^{(curr)} * \mu_i$ then      ▷ In this case the M/M/c is instable
18:         $\rho^{(overload)} \leftarrow \rho^{(overload)} + \lambda_i - \mathsf{c}^{(curr)} * \mu_i$
19:         $u^{(curr)} \leftarrow 1.0$
20:     else      ▷ Use M/M/c based model
21:         $u^{(curr)} \leftarrow \min(\tilde{u} + \rho^{(overload)}, 1.0)$      ▷ Get $\tilde{u}$ from Proposition 2
22:         $\rho^{(overload)} \leftarrow \rho^{(overload)} - (u^{(curr)} - \rho^{(overload)})$
23:     end if
24:     if $\left| \frac{u^{(curr)}}{\hat{u}} - 1 \right| > \sigma$ then      ▷ Use (20) with $u_j = u^{(curr)}$
25:         $\mathsf{c}^{(next)} \leftarrow \left\lceil \mathsf{c}^{(curr)} \frac{u^{(curr)}}{\hat{u}} \right\rceil$      ▷ Use (21) with $(curr)$ as scaling interval $j$
26:     else
27:         $\mathsf{c}^{(next)} \leftarrow \mathsf{c}^{(curr)}$
28:     end if
29:     return $\mathsf{c}^{(next)}, \rho^{(overload)}$
30: end procedure
```

Using this model of an single elastic $(s.e.)$ substrate node, we interpret their capacities, the demands of the to-be-placed VNFs and further extend the capacity and demand functions with an additional type:

$$\begin{aligned} &\forall t \in [0, T_R], \forall v_i \in V_R, \forall s_j \in V_S, \mathcal{T}_S(s_j) = (s.e.) : \\ &\qquad d_R^{(s.e.)} = \text{MODELHPABEHAVIOR}(v_i, \lambda_i, \mu_i, s_j :: [\mathsf{c}_0, \mathsf{c}_{min}, \mathsf{c}_{max}, \hat{u}, t^{(sca)}, \sigma, N^{(sca)}]) \\ &\qquad d_R^{(s.e.)}(t, v_i, s_j) = f^{(pod)}(t, v_i, s_j) \end{aligned} \tag{23}$$

$$\forall s_i \in V_S, \mathcal{T}_S(s_i) = (s.e.) : \qquad c_S^{(s.e.)}(s_i) = s_i :: \mathsf{c}_{max} \tag{24}$$

The demand function of a single VNF, to-be-placed on an elastic node is defined as the output of Algorithm 3, being either the pod count trajectory $f^{(pod)}$ or the invalid character $\perp$. Assuming a strict QoS approach for each VNF, we allow the placement of a set of VNFs, if their sum of demand functions (i.e., their pod count trajectories) would never go above the maximal pod count c_{max} of an elastic node s_i. Furthermore, if a VNF alone would exceed c_{max}, the containing VNF set is considered not placeable.

The above interpretation of elastic resource capacities and demands, defines the admission control function for computation capacity managed by Kubernetes HPA. The effect of the incoming requests are modeled based on the infrastructure's configuration, and this is used to compare the predicted dynamic demand to the allocated infrastructure capabilities. Further extending the capacity constraint (18) (which is an extension of (6) of Formulation 1) we get:

$$\forall s_i \in V_S, r \subseteq V_R : c_S(r, s_i) = \begin{cases} 1, & \text{if } \sum_{v \in r} d_R^{(s.f.)}(v) \leq c_S^{(s.f.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (s.f.) \\ 1, & \text{if } \sum_{v \in r} d_R^{(s.f.)}(v) \leq c_S^{(a.f.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (a.f.) \\ 0, & \text{if } \exists v \in r : d_R^{(s.e.)}(t, v, s_i) = \perp \text{ and } \mathcal{T}_S(s_i) = (s.e.) \\ 1, & \text{if } \forall t \in [0, T_R] : \sum_{v \in r} d_R^{(s.e.)}(t, v, s_i) \leq c_S^{(s.e.)}(s_i) \text{ and } \mathcal{T}_S(s_i) = (s.e.) \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

In addition to modeling the capacity of an elastic node, the defined demand function enables us to refine the cost model of a VNF deployment on elastic substrate. If the cost of running a pod of VNF v_i on elastic node s_j for one time unit is $p^{(pod)}(v_i, s_j)$, then the total cost of placing the VNF here for the whole service lifetime T_R can be calculated:

$$v_i \in V_R, s_j \in V_S, \mathcal{T}_S(s_j) = (s.e.), T_R \in \mathbb{R}^+ :$$
$$p_{V_R}(T_R, v_i, s_j) = p^{(pod)}(v_i, s_j) \int_0^{T_R} d_R^{(s.e.)}(\tau, v_i, s_j) d\tau \tag{26}$$

The elastic substrate node model integrates into our heterogeneous infrastructure model, enabling the CSP to expose its detailed view of its dynamic infrastructure towards an OTT SP if necessary. The OTT SP's orchestration algorithm can utilize this information to make VNF placement decisions and accurate cost estimations using pod count trajectory prediction. Implementing Algorithm 3 and Equations (25) and (26) inside the MAPONENF(e_i, v_i) function, enables our greedy heuristic framework to serve as the OTT SP's orchestration algorithm over the CSP's hybrid detailed-abstract infrastructure resources.

7. Practical Exploitation

In this section we summarize how our results could be integrated with the referred 5G prototype systems [8,19], and how they can be used for the example *remote security camera analytics and alerts* IoT use case. We demonstrate the running time of Algorithm 1, and compare it to the times required by other components of our prototype system. Furthermore, we present numeric examples for applications of Theorem 1, the resource contraction problem solved by Algorithm 2 and comparing the elastic resource model of Algorithm 3 to measurement. Figure 4 positions the presented algorithms in a general 5G-enabled IoT framework. Finally, we briefly discuss the challenges of the aggregate elastic resource type.

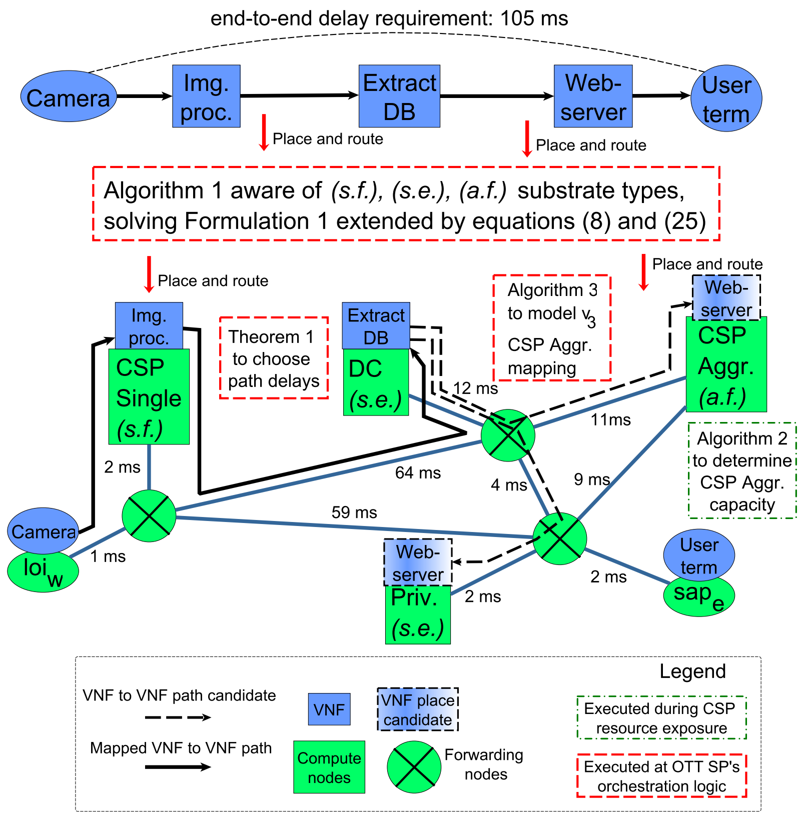
end-to-end delay requirement: 105 ms
Camera
Img. proc.
Extract DB
Web-server
User term
Place and route
Place and route
Algorithm 1 aware of (s.f.), (s.e.), (a.f.) substrate types, solving Formulation 1 extended by equations (8) and (25)
Place and route
Place and route
Img. proc.
CSP Single (s.f.)
Theorem 1 to choose path delays
Extract DB
DC (s.e.)
Algorithm 3 to model v3 CSP Aggr. mapping
Web-server
CSP Aggr. (a.f.)
Algorithm 2 to determine CSP Aggr. capacity
12 ms
11ms
2 ms
64 ms
4 ms
9 ms
59 ms
Camera
loi w
1 ms
Web-server
Priv. (s.e.)
2 ms
2 ms
User term
sap e
Legend
VNF to VNF path candidate
Mapped VNF to VNF path
VNF
VNF place candidate
Compute nodes
Forwarding nodes
Executed during CSP resource exposure
Executed at OTT SP's orchestration logic

Figure 4. Illustration of applying the presented results in a 5G prototype architecture for the remote security camera analytics and alerts IoT application.

7.1. Result Application Examples

As shown by Figure 4, the core service graph embedding is executed by the OTT SP's orchestration logic, which must guarantee the end-to-end delay requirements, and be aware of the various substrate node types of the underlying heterogeneous infrastructure. As it was demonstrated, our contribution enables Algorithm 1 to solve and extended version of Formulation 1 including the delay constraint of (8) and computation capacity constraints in (25). Our results empower existing and future 5G architecture deployments to realize various IoT services, spanning wide geographic regions over software-controlled, abstract and dynamic infrastructures provided by multiple communication and computation entities.

7.1.1. Real-World Prototype Experiment

We have conducted experiments in a real-world sandbox environment, where the computation nodes managed by Algorithm 1 are distributed across multiple European locations [19]. The proof-of-concept (PoC) 5G architecture prototype integrates Virtual Infrastructure Managers (VIMs), which expose their computation resources for an OTT SP's orchestrator running Algorithm 1. Comparing our PoC to Figure 4, the role of a CSP is played by European Internet Service Providers, while the computation infrastructure is provided by the participating institutions of the research project. We measured the deployment time of two types of Robot Control services with high reliability constraints, i.e., having the robot's control logic provided by two independently deployed VNFs. The details of the sandbox environment, prototype implementation and the experiment scenarios are published in our earlier work [19], here we only summarize the relevant findings.

Table 6 shows the experiment results in terms of timing, for both application types. The difference in the services is their communication underlay, which from the service placement algorithm's perspective means different number of components. In both cases the running time of Algorithm 1 is orders of magnitudes smaller than the deployment time required by the underlying VIMs for starting and configuring the to be instantiated VNFs. In the IP routing case more components of the whole PoC architecture are involved in creating the service, this is the result of the significantly higher overall time.

Table 6. Sandbox experiments of our prototype implementation of Algorithm 1 [19].

	Robot Control (SDN Routing)	**Robot Control (IP Routing)**
Running time of Algorithm 1	0.25 s	0.32 s
VNF deployment at the VIMs	110.23 s	217.96 s
Overall time	112.67 s	219.2 s

We conclude that the computation requirement of Algorihtm 1 do not cause any bottlenecks in the PoC, so enriching its features with the elastic resource modeling and infrastructure aggregating of the well-scaling algorithms presented in this paper is possible. Our current paper complements this PoC with firm theoretical results to propose foundations for modeling the infrastructure resources.

7.1.2. Application of Theorem 1

Our result on greedy heuristic bound for end-to-end delay, can be used to design the step-by-step bounds for guaranteed constraint approximation performance. The OTT SP's orchestration logic can use the conditions of Theorem 1 to make a path-delay-aware decision while choosing the routing path for individual service graph connections. Path options are illustrated by dashed arrows in Figure 4. The following numerical example demonstrates how such bounds can be designed.

The expression for the bounds product $B^{(dist)}B^{(req)}$ of Theorem 1 can be rearranged to express the estimation of ϵ:

$$\epsilon \geq n + \alpha - \frac{(n+1)B^{(dist)}B^{(req)}}{D^{(req)}(\alpha D^{(req)} + d_{s_0t})}. \tag{27}$$

To numerically calculate the example, we take a 10-long service chain as the service graph, i.e., $n = 10$. Taking a permissive approach for designing the bounds (i.e., $B^{(dist)}$and $B^{(req)}$), and preserving their dependency on problem input parameters, we may take the $\frac{\sqrt{3}}{2}$ of their allowed intervals, as shown in Table 7.

Table 7. Numerical example for applying Theorem 1 for greedy algorithm design.

Algorithm Parameter	**Chosen Formula**	**Numerical Value ($n = 10, \alpha = 0.9$)**
$B^{(dist)}$	$\frac{\sqrt{3}}{2}\frac{n+\alpha}{n+1}D^{(req)}$	$0.85D^{(req)}$
$B^{(req)}$	$\frac{\sqrt{3}}{2}(\alpha D^{(req)} + d_{s_0t})$	$0.78D^{(req)} + 0.87d_{s_0t}$
ϵ	$\frac{1}{4}(n+\alpha)$	2.72

Substituting these bounds to Equation (27) and making the simplifications we get actual bound for the approximation value ϵ, shown in Table 7. Having the bound for ϵ we can choose the value of α from its allowed interval $\alpha = 0.9 \in (\frac{14}{43}, 1]$. This enables us to calculate the numeric value for all other algorithm parameters, presented in the last column of Table 7.

Note that the shown $(1+\epsilon)$-approximation does not utilize the backtracking capability of the greedy heuristic, which drastically lowers the end-to-end delay augmentation or meets the requirement. The shown method formalizes the design process of greedy end-to-end heuristics, to tackle the trade-off between computation complexity and constraint violation. This result enables us to control the computational impact of Algorithm 1 in the whole orchestration process.

7.1.3. Resource Contraction Example Using Algorithm 2

To facilitate the VNF placement on the aggregate fixed $(a.f.)$ type substrate nodes, the CSP wishes to aggregate the capacities of its servers, and show an abstract view of the computation capacities to the OTT SP's orchestration logic. The *CSP Aggr.* compute node of Figure 4 illustrates this scenario. The VNF placement logic uses the $(a.f.)$ case of (25) to decide whether a VNF can be placed on the aggregate substrate node. The following numerical example shows the $\frac{1}{2}$-optimal solution of the NP-hard problem of Formulation 3 by our proposed approximation algorithm shown in Algorithm 2.

For simplicity we omit the capacity and demand functions from the inputs of the related problems and algorithms, and only list the substrate node capacities, VNF demands in the $\texttt{CAP}$, $\texttt{RCP}$and $\textsc{aprx}\text{RCP}$ arguments. We take the base VNF set $\mathcal{R}^{(vnfs)}$ with capacity demands $Dem = \{1,2,4\}$ and to-be-aggregated substrate nodes V_S' with capacities $Cap = \{2,3,5,7,8\}$. First, we estimate $OPT\{\texttt{RCP}(Cap, Dem)\}$, which is obviously smaller than 20, as 5 times the VNF with demand 4 could be allocated on the aggregate node, i.e., $\texttt{CAP}(\{20\},\{4,4,4,4,4\})$ is true, but could not be allocated on the hidden nodes, i.e., $\texttt{CAP}(Cap,\{4,4,4,4,4\})$ is false. Our proposed resource contraction algorithm $\textsc{aprx}\text{RCP}(Cap, Dem)$, shown in Algorithm 2, in its first iteration with $d_{\max} = 4$, calculates $A_0 = 2+4+5 = 11$ for the big capacities $\{5,7,8\}$. In the second iteration for the small capacities, it calculates $A_1 = \textsc{aprx}\text{RCP}(\{2,3\},\{1,2\}) = 3$. So the aggregate capacity $\textsc{aprx}\text{RCP}(Cap, Dem) = A_0 + \min\{d_{\max} - 1, A_1\} = 14$, which is at least $\frac{14}{19}$-approximation of the optimal solution.

In conclusion, the CSP can efficiently show its resources meeting the essential property for abstraction, as described in Remark 1, which view can be quickly generated.

7.1.4. Comparison of Algorithm 3 to Measurement

Finally, our results on modeling Kubernetes HPA, the OTT SP, running Algorithm 1, can estimate the trajectory of VNF replication states for the IoT service's lifetime in response to user traffic. The trajectory estimation is presented in Algorithm 3 and should be implemented in the OTT SP's orchestration logic, making placement decisions based on (25). This is illustrated by the placements of *Extract DB* VNF and *Webserver* VNF to single elastic $(s.e.)$ substrate nodes on Figure 4.

Figure 5 shows the predicted pod count trajectory and its comparison to the behavior of a real Kubernetes cluster deployment managed by HPA. The experiment uses a webserver VNF packaged in a pod, receiving independent, identically and exponentially distributed user requests with constant arrival ($\lambda_i = 10 \frac{request}{min}$) and processing intensity ($\mu_i = 1 \frac{request}{min}$) for an hour.

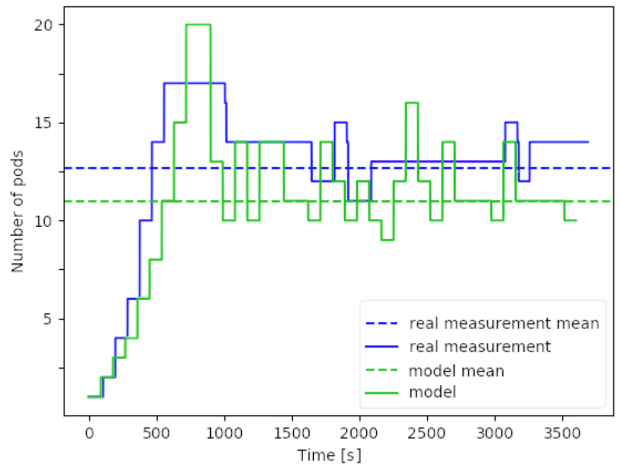

Figure 5. Validative example of single elastic resource model by Algorithm 3 compared to a real Kubernetes Horizontal Pod Autoscaler (HPA) pod count trajectory.

The HPA's scaling interval $t^{(sca)}$ is set to 1 min, with targeted scaling metric $\hat{\mathfrak{u}} = 0.5$, scaling tolerance $\sigma = 0.1$ and pod limits $\mathtt{c}_{min} = 1$, $\mathtt{c}_{max} = 20$. The model utilizes the calculated arrival and processing rate of the webserver VNF during the current scaling interval. So that, not only a fixed pair of arrival rate λ_i and processing rate μ_i is given (as shown in Algorithm 3), but the realization of the randomized processes is used to calculate the local rates. This approach slightly generalizes the algorithm to provide a reasonable comparison to the measurement, as the Kubernetes HPA is not aware of the exact rates used for generating the arrival and processing times. These estimated rates are used as the input for the NEXTHPAPODCOUNT() function to predict the utilization metric, and thus predict the number of active pods for the next scaling interval. The experiment starts with a single running pod, when both the model and the measurement initially over-provisions with very similar extent and timing. Both trajectories follow the slight oscillation around the equilibrium pod count of 11–13, resulting in at most $-/+15\%$ deviation

Using this model the OTT SP can make a well-informed decision for deploying its IoT service over the CSP's elastic and heterogeneous infrastructure.

7.2. Aggregate Elastic Substrate Node Type

Designing a model for aggregate elastic $(a.e.)$ infrastructure node types is extremely challenging, as modeling elasticity requires various parameters describing the dynamic behavior, while aggregation abstracts the lower level infrastructure details. For instance, aggregating the view of multiple Kubernetes HPA nodes using the resource contraction problem (RCP) would require allocating a fix capacity to each of the to-be-hidden clusters. This goes against preserving the dynamic nature of the underlying nodes. On the other hand, describing the aggregate elastic node's capabilities using well-studied queuing systems, such as the M/M/c queue, would require crude simplifications of infrastructure characteristics.

This is a possible future research direction in infrastructure modeling as dynamicity increasingly gains momentum in cloud computing, while network management of edge resources is already challenging, calling for higher computing and communication resource abstractions.

8. Conclusions

We have identified a gap between the rapid evolution of virtualization-based infrastructure technologies and the essentially unchanged substrate models of service placement algorithms. Distributed computing solutions, such as Mobile Edge Computing and 5G utilize increasingly capable infrastructure technologies for addressing quickly changing user demands, and scaling up to the immense sizes of IoT networks. On the contrary, the service component placement algorithms, managing service deployment on top of these architectures, only consider fixed substrate capacities.

Primarily, we have presented our resource aggregation model to support the scalability of network management algorithms, and help preserving sensitive infrastructure information during inter-provider control-plane communication. Secondly, we have used our classic queuing theory results to build an elastic resource model for the auto-scaling mechanism of the wide-spread container management system, Kubernetes. Our results on infrastructure modeling empower existing 5G prototype architectures to place delay-constrained IoT service components on highly dynamic and abstract resource views. In addition, we demonstrated how scalable service placement algorithms can meet path delay requirements by making greedy decisions, while optimizing for independent metrics. Our results integrate with placement algorithms using elementary service-to-resource mapping decisions, enabling their easy deployments already in the proof-of-concept phase. All of our results are complemented with rigorous mathematical theorems whose proofs have been presented in detail.

Finally, we have showcased the practical applicability of our theorems and algorithms to demonstrate what roles our results play for realizing the *remote security camera analytics and alerts* IoT application across wide geographic areas. The infrastructure resources provided by multiple CSPs and CPs are used by customers facing OTT SPs, who provide IoT services. All players of these complex business relations benefit from higher resource abstraction, accurate modeling of dynamic resources and efficient service placement algorithms.

The computation and communication infrastructure for realizing various IoT applications is becoming increasingly capable to keep up with the user requirements for high service quality. In emerging modern networks, the management algorithms not only have to handle communication resources, but the coordinated management of computation needs to be blended in. The contributions of this paper are believed to be essential for effective abstract resource information exchange and detailed modeling of elastic resources. We hope that our results could serve as a basis for such communication protocol definitions and standardized data models for multi-domain, heterogeneous 5G infrastructures.

Author Contributions: Conceptualization, B.N. and B.S.; formal analysis, B.N.; writing—original draft preparation, B.N. and B.S.; writing—review and editing, B.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Proof of Theorem 1. For easier notation and without loss of generality, we index the substrate nodes by the index of the hosted VNFs $\forall v_i \in V_R, \mathfrak{V}(v_i) = s_i$, with potential substrate aliases for VNF colocation (i.e., $s_1 = s_2, \mathfrak{V}(v_1) = \mathfrak{V}(v_2)$).

As the delay distances between any two substrate nodes $s_i, s_j \in V_S$ is pre-calculated, the following fraction could be calculated in each greedy step of the algorithm. The numerator is a lower estimation of the total used delay (if v_{i+1} is going to be hosted on the specific substrate node s_{i+1}) until the end of the whole service graph path $\mathfrak{V}(v_{n+1}) = s_{n+1} = t$. This expression can be interpreted as the sum of path delay on a feasible path for request edge $(v_i, v_{i+1}) \in E_R$, and the host candidate s_{i+1}'s distance to t. The denominator is an upper estimation of the numerator as it expresses the local remaining delay budget, if $\mathfrak{V}(v_{i+1}) = s_{i+1}$.

$$F_i = \frac{d^{(fp)}_{s_i s_{i+1}} + d_{s_{i+1}t}}{D^{(req)} - \sum_{j=0}^{i-1} d^{(fp)}_{s_j s_{j+1}}}, \forall v_i \in V_R = \{v_0, v_1, \dots v_n\}, \mathfrak{V}(v_i) = s_i \tag{A1}$$

The fraction F_i is always positive, as the numerator represents distances and the denominator is the remaining delay budget $D^{(s_i)}$ at the given greedy step. The delay budget at the beginning of the mapping is the end-to-end delay requirement $D^{(req)}$, when the current VNF is v_0 with fixed location s_0. Let us take the commonly known inequalities between the harmonic and arithmetic means of the $n + 1$ fractions $F_i \; \forall v_i \in \{v_0, v_1, \dots v_n\}$:

$$Harmonic(F_i) = \frac{n+1}{\sum\limits_{i=0}^{n} \frac{D^{(req)} - \sum_{j=0}^{i-1} d^{(fp)}_{s_j s_{j+1}}}{d^{(fp)}_{s_i s_{i+1}} + d_{s_{i+1}t}}} \leq \frac{\sum\limits_{i=0}^{n} \frac{d^{(fp)}_{s_i s_{i+1}} + d_{s_{i+1}t}}{D^{(req)} - \sum_{j=0}^{i-1} d^{(fp)}_{s_j s_{j+1}}}}{n+1} = Arithmetic(F_i) \tag{A2}$$

Upper bounding the right side expression by replacing the internal denominator by the delay requirement bound $B^{(req)}$, we get common denominator for each element. Furthermore, replacing the delay distance bound $d_{s_{i+1}t} \leq d_{s_0 t}$ as stated in the conditions of the theorem, each element is strictly increased as shown in (A3). Upper bounding each feasible path delay by $\alpha D^{(req)}$, and simplifying by $(n+1)$ we get the final upper bound.

$$Arithmetic(F_i) \leq \frac{\sum_{i=0}^{n} d_{s_i s_{i+1}}^{(fp)} + d_{s_0 t}}{(n+1)B^{(req)}} \leq \frac{(n+1)\left(\alpha D^{(req)} + d_{s_0 t}\right)}{(n+1)B^{(req)}} = \frac{\alpha D^{(req)} + d_{s_0 t}}{B^{(req)}} \tag{A3}$$

Secondly, we lower bound the harmonic mean of the fractions, by applying the trivial lower bound for the feasible path delay $d_{s_i s_{i+1}}^{(fp)} \geq d_{s_i s_{i+1}}$. This decreases the denominator in each element of the sum, which increases the total in the main denominator, resulting in a lower main fraction value. The second lower estimation is done by replacing each element's denominator by the lower bound on the used substrate node's distance $B^{(dist)}$, resulting in a common denominator:

$$Harmonic(F_i) \geq \frac{n+1}{\sum_{i=0}^{n} \frac{D^{(req)} - \sum_{j=0}^{i-1} d_{s_j s_{j+1}}^{(fp)}}{d_{s_i s_{i+1}} + d_{s_{i+1} t}}} \geq \frac{(n+1)B^{(dist)}}{\sum_{i=0}^{n} \left(D^{(req)} - \sum_{j=0}^{i-1} d_{s_j s_{j+1}}^{(fp)}\right)} \geq \dots \tag{A4}$$

Next, we increase the denominator by discarding negative values from each element of the sum, moreover, for each index i only the last element of the inner sum (with the highest index j) is kept. Rearranging the sum, we get the total used path delay until VNF v_n, which equivalently can be rewritten as shown in (A5). As the feasible path delay is upper bounded by $\alpha D^{(req)}$ for each step, we get the last lower estimation.

$$\dots \geq \frac{(n+1)B^{(dist)}}{(n+1)D^{(req)} - D_n^{(used)}} = \frac{(n+1)B^{(dist)}}{(n+1)D^{(req)} - D_{n+1}^{(used)} + d_{s_n t}^{(fp)}} \geq \frac{(n+1)B^{(dist)}}{(n+1+\alpha)D^{(req)} - D_{n+1}^{(used)}} \tag{A5}$$

Meeting the two sides of the estimations we get

$$\frac{(n+1)B^{(dist)}}{(n+1+\alpha)D^{(req)} - D_{n+1}^{(used)}} \leq \frac{\alpha D^{(req)} + d_{s_0 t}}{B^{(req)}} \tag{A6}$$

which can be rearranged to a general upper bound of the total used path delay $D_{n+1}^{(used)}$

$$D_{n+1}^{(used)} \leq (n+1+\alpha)D^{(req)} - \frac{(n+1)B^{(dist)}B^{(req)}}{\alpha D^{(req)} + d_{s_0 t}}. \tag{A7}$$

Substituting the expression as stated for the product of the bounds $B^{(dist)}B^{(req)}$, we upper bound the right-hand side and get the main statement of the theorem for $(1+\epsilon)$-approximation. The upper limits stated for the bounds individually can be verified by substituting them into (A6). □

Appendix B

Proof of Theorem 2. First of all, we state and prove some lemmas, to support the claims of the theorem (Thanks for Márton Zubor, PhD for his essential help with this proof).

Lemma A1. *Let VNF $v_{\max} \in \mathcal{R}^{(vnfs)}$ be with maximum demand $d_{\max} := \max_{d_R^{(s.f.)}}(\mathcal{R}^{(vnfs)})$, and let substrate node $s \in V_S$ be with capacity $c_S^{(s.f.)}(s) \geq d_{\max}$. Furthermore, let request multiset be $(V_R', m_{R'})$, $V_R' \subseteq \mathcal{R}^{(vnfs)}$, such that $dem\_sum((V_R', m_{R'})) = \sum_{v_i \in V_R'} d_R^{(s.f.)}(v_i) m_{R'}(v_i) > c_S^{(s.f.)}(s)$.*

Then, $\exists (V_R'', m_{R''}) \subseteq (V_R', m_{R'})$ so that

$$\max\left\{\left\lfloor \frac{c_S^{(s.f.)}(s)}{2} \right\rfloor + 1,\ c_S^{(s.f.)}(s) - d_{\max} + 1\right\} \leq dem\_sum((V_R'', m_{R''})) \leq c_S^{(s.f.)}(s) \tag{A8}$$

Proof of Lemma A1. Choose elements of $(V_R'', m_{R''})$ from $(V_R', m_{R'})$ in non-increasing order (starting with highest demand VNF), until the inequality $dem\_sum((V_R'', m_{R''})) \leq c_S^{(s.f.)}(s)$ holds. Then clearly (even if $v_{\max}$ was the only chosen element) $c_S^{(s.f.)}(s) - d_{\max} < dem\_sum((V_R'', m_{R''}))$ holds. As the values are integers, adding 1 to the right-hand side, the inequality is reached.

Without loss of generality we can index the ordered list of $(V_R{'}, m_{R'})$, where $M = size((V_R{'}, m_{R'}))$: $d_R^{(s.f.)}(v_1) \geq d_R^{(s.f.)}(v_2) \geq \cdots \geq d_R^{(s.f.)}(v_M)$. We define the notation for the multisets with listed elements in this order and their corresponding demand sum as

$$(V_R{}^{(i)}, m_{R^{(i)}}) = \{v_1, \ldots v_i\}, \qquad \forall i = 1 \ldots M \tag{A9}$$

$$D_i = dem\_sum((V_R{}^{(i)}, m_{R^{(i)}})), \qquad \forall i = 1 \ldots M \tag{A10}$$

Since $d_{\max} \leq c_S^{(s.f.)}(s)$, we can choose and index $k \in \{1 \ldots M-1\}$ such that

$$d_R^{(s.f.)}(v_1) = D_1 < \cdots < D_k \leq c_S^{(s.f.)}(s) < D_{k+1} < \cdots < D_M = dem\_sum((V_R{}^{(M)}, m_{R^{(M)}})) \tag{A11}$$

Looking at the interesting part of these inequalities

$$D_k \leq c_S^{(s.f.)}(s) < D_{k+1} = D_k + d_R^{(s.f.)}(v_{k+1}) \tag{A12}$$

$$c_S^{(s.f.)}(s) - d_R^{(s.f.)}(v_{k+1}) < D_k \tag{A13}$$

From the initial indexing $d_R^{(s.f.)}(v_{k+1}) \leq d_R^{(s.f.)}(v_k)$ and we get $d_R^{(s.f.)}(v_{k+1}) \leq D_k$ as the demand of v_k is contained in D_k. Adding this to the previous equation

$$c_S^{(s.f.)}(s) < 2D_k \tag{A14}$$

With using the chosen index k as the multiset to be found $(V_R{''}, m_{R''})$, we get the remaining term of the statement of the lemma:

$$\left\lfloor \frac{c_S^{(s.f.)}(s)}{2} \right\rfloor + 1 \leq D_k = dem\_sum((V_R{''}, m_{R''})) \tag{A15}$$

□

Lemma A2. *Let A_0 be as defined in line 7 of Algorithm 2, then the following inequalities hold:*

$$\frac{cap\_sum(V_S'^{(big)})}{2} \leq A_0 \leq OPT\{\mathtt{RCP}(V_S'^{(big)}, \mathcal{R}^{(vnfs)}, c_S^{(s.f.)}, d_R^{(s.f.)})\} \tag{A16}$$

Proof of Lemma A2. The right inequality directly follows from the definition of A_0, because for all substrate nodes with big capacity $s_i \in V_S'^{(big)}$ the inequality $\frac{c_S^{(s.f.)}(s_i)}{2} \leq \max\left\{\left\lfloor \frac{c_S^{(s.f.)}(s_i)}{2} \right\rfloor + 1,\ c_S^{(s.f.)}(s_i) - d_{\max} + 1\right\}$.

The second inequality means we can solve the Capacity Allocation Problem for all request multisets $\forall V_R' \subseteq \mathcal{R}^{(vnfs)}, \forall (V_R', m_{R'})$, where $dem\_sum((V_R{'}, m_{R'})) \leq A_0$ because the condition of the essential property in (16) are met, i.e., the sum of demands is lower than the capacity of the aggregate node created from the big capacity substrate nodes $V_S'^{(big)}$. For proving the inequality, we need to provide a solution for the Capacity Allocation Problem for all VNF multisets $(V'_R, m_{R'})$ onto substrate nodes $V_S'^{(big)}$.

If there is any substrate node $s_i \in V_S'^{(big)}$, which can fit all of the VNFs, i.e., $dem\_sum((V_R{}', m_{R'})) \leq c_S^{(s.f.)}(s_i)$, then the allocation is given. Otherwise, in this situation, according to Lemma A1 we can choose some VNF multiset $(V_R{}'', m_{R''}) \subseteq (V_R{}', m_{R'})$, which completely fits to a substrate node $s_i \in V_S'^{(big)}$. Repeat this step by discarding the used substrate node s_i and the allocated VNFs $(V_R{}'', m_{R''})$, until a complete mapping of $(V_R{}', m_{R'})$ is found. Indirectly suppose that this iterative method fails, which means there is a multiset of VNFs $(V_R{}^*, m_{R^*})$ for which $dem\_sum((V_R{}^*, m_{R^*})) > A_0$, but this contradicts the initial condition of fitting to the aggregate node. □

Lemma A3. *If substrate node set $V'_S \subseteq V_S$ is disjoint union of $V_S^{(0)}$ and $V_S^{(1)}$, then (the capacity and demand functions are omitted for readibility)*

$$OPT\{\texttt{RCP}(V'_S, \mathcal{R}^{(vnfs)})\} \leq OPT\{\texttt{RCP}(V_S^{(0)}, \mathcal{R}^{(vnfs)})\} + OPT\{\texttt{RCP}(V_S^{(1)}, \mathcal{R}^{(vnfs)})\} \tag{A17}$$

Proof of Lemma A3. Let

$$A' = OPT\{\texttt{RCP}(V'_S, \mathcal{R}^{(vnfs)})\}$$
$$A^{(0)} = OPT\{\texttt{RCP}(V_S^{(0)}, \mathcal{R}^{(vnfs)})\}$$
$$A^{(1)} = OPT\{\texttt{RCP}(V_S^{(1)}, \mathcal{R}^{(vnfs)})\}$$

Then the left side of the inequality means we can solve $\texttt{CAP}(V'_S, (V'_R, m_{R'}), c_S^{(s.f.)}, d_R^{(s.f.)})$ for any request multiset $\forall V'_R \subseteq \mathcal{R}^{(vnfs)}, \forall (V'_R, m_{R'})$, where $dem\_sum((V_R{}', m_{R'})) \leq A'$. While on the right side, if we take the same VNF multiset $(V_R{}', m_{R'})$ we can solve the corresponding CAP-s if $(V_R{}', m_{R'})$ has a special 2-set partition $(V_R{}^{(0)}, m_{R^{(0)}})$ and $(V_R{}^{(1)}, m_{R^{(1)}})$ so that $dem\_sum((V_R{}^{(0)}, m_{R^{(0)}})) \leq A^{(0)}$ and $dem\_sum((V_R{}^{(1)}, m_{R^{(1)}})) \leq A^{(1)}$. These request sets can be collected from the solution of the CAP for the initial substrate set V'_S as $V_S^{(0)}$ and $V_S^{(1)}$ are given.

As the $OPT\{\texttt{RCP}\}$ function is subadditive for the substrate graph sets (i.e., adding new substrate nodes does not necessarily increase $OPT\{\texttt{RCP}\}$ due to e.g., not being able to map more VNFs anyway), and the CAPproblems are feasible, the inequality follows. □

Now we return to the proof of the main statement. Note that the demand and capacity functions are omitted from the input of $OPT\{\texttt{RCP}\}$ and APRXRCP as they do not change during the proof. We prove $OPT\{\texttt{RCP}(V'_S, \mathcal{R}^{(vnfs)})\} \leq 2 \cdot \text{APRXRCP}(V'_S, \mathcal{R}^{(vnfs)})$ by induction on the number of elements in $\mathcal{R}^{(vnfs)}$.

We take the partition of the substrate nodes V'_S into big capacity nodes $V_S'^{(big)}$ and small capacity nodes $V_S'^{(small)}$ as it is used in Algorithm 2. From Lemma A2 and from (15) of Formulation 3, respectively we get

$$cap\_sum(V_S'^{(big)}) \leq 2A_0 \quad \Leftarrow \text{(Lemma A2)} \tag{A18}$$
$$OPT\{\texttt{RCP}(V_S'^{(big)}, \mathcal{R}^{(vnfs)})\} \leq cap\_sum(V_S'^{(big)}) \quad \Leftarrow (15) \tag{A19}$$
$$OPT\{\texttt{RCP}(V_S'^{(big)}, \mathcal{R}^{(vnfs)})\} \leq 2A_0 \tag{A20}$$

Solving the RCP for the small substrate nodes $V_S'^{(small)}$ is clearly the demand of the smallest fitting VNF (reminiscent of line 11 of Algorithm 2):

$$OPT\{\texttt{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)})\} = \min\left\{d_{\max} - 1,\ OPT\{\texttt{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\})\}\right\} \tag{A21}$$

where VNF $v_{\max} \in \mathcal{R}^{(vnfs)}$ is with maximum demand $d_{\max} := \max_{d_R^{(s.f.)}}(\mathcal{R}^{(vnfs)})$. By the inductive assumption on the number of VNFs we know

$$OPT\{\texttt{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\})\} \leq 2\textsc{aprx}\text{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\}) \tag{A22}$$

Substituting it into the previous equation we get the inequality

$$OPT\{\texttt{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)})\} \leq 2\min\left\{d_{\max} - 1,\ \textsc{aprx}\text{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\})\right\} \tag{A23}$$

Adding together the inequalities (A20) and (A23), we get the recursive return value of $\textsc{aprx}\text{RCP}$ on the right side:

$$\begin{aligned} &OPT\{\texttt{RCP}(V_S'^{(big)}, \mathcal{R}^{(vnfs)})\} + OPT\{\texttt{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)})\} \leq \\ &2A_0 + 2\min\left\{d_{\max} - 1,\ \textsc{aprx}\text{RCP}(V_S'^{(small)}, \mathcal{R}^{(vnfs)} \setminus \{v_{\max}\})\right\} \end{aligned} \tag{A24}$$

Finally, applying Lemma A3 to the disjoint sets of $V_S'^{(big)}$ and $V_S'^{(small)}$ we get the lower estimation by the optimal value on the left side:

$$OPT\{\texttt{RCP}(V_S', \mathcal{R}^{(vnfs)})\} \leq 2 \cdot \textsc{aprx}\text{RCP}(V_S', \mathcal{R}^{(vnfs)}) \tag{A25}$$

□

Appendix C

Proof of Theorem 3. The condition of $c\mu \leq \lambda$ ensures the stability of the system, i.e., all of the subsequently defined random variables have first moments [32,33].

We build our results on the foundational analytical work on M/M/c models presented in [33]. We restate some of their results and definitions using the notations of our paper to prove the theorem. We take the definition of the M/M/c busy period T_k as the time until the first occasion the system decreases the number of active pods from k to $k-1$. It is formalized in Definition A1.

Definition A1 (Busy period T_k of M/M/c [33])**.** *Let $k \in \{1, 2, \ldots c\}$ be the number of busy pods in an M/M/c system, and let $T_k \in \mathbb{R}^+$ be a random variable:*

$$\begin{aligned} T_k = min[t\ :\ &k \text{ jobs in the system at time } 0^+, \\ &k-1 \text{ jobs in the system at time } t]. \end{aligned}$$

Using the definition of T_k we formally define the total busy time Θ_c used in Theorem 3 as the sum of all busy times. Due to the memoryless property of the M/M/c system, the subsequent values of the independent random variables T_k can be summed.

Definition A2 (Total busy time Θ_c of M/M/c)**.** *The random variable of the total busy time Θ_c measures the length of time until the first idle moment of an M/M/c system starting from c busy servers.*

$$\Theta_c = \sum_{k=1}^{c} T_k$$

Following the idea of [33] for the definition of T_k, we analogously define the random variable of the single weighted busy period B_k. A demonstration is shown in Figure A1 and the formal definition in Definition A3.

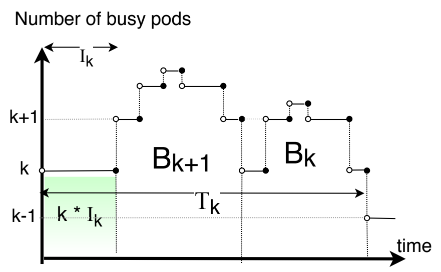

Figure A1. Illustration for the recursive definition of B_k.

Definition A3 (Weighted single busy period B_k of M/M/c)**.** *Let I_k be the random variable of inter-event time of the aggregate Poisson process of an M/M/c system with*

- *k Poisson processes (pods) serving requests with rate μ and*
- *the arrival Poisson process with rate λ.*

Denote the probabilities of an arrival and departure of such aggregated Poisson process by $p_k^{(arr)}$ and $p_k^{(dep)}$, respectively. Let B_k, $k \in \{1, 2, \dots c\}$ be the weighted total load of an M/M/c system while first time reducing the number of currently active pods from k to $k-1$:

$$B_k = \begin{cases} = \begin{cases} kI_k + B_{k+1} + B_k, & \text{with probability} \quad p_k^{(arr)} \\ kI_k, & \text{with probability} \quad p_k^{(dep)} \end{cases} & \text{if } 1 \leq k < c \\ = cT_c & \text{if } k = c. \end{cases}$$

At Definition A3 we recursively use the mixture distribution of random variables, defined by the arrival and departure events. For the case of $1 \leq k < c$ the first event is either a departure or arrival after I_k time has passed. The area under the chart until this event is kI_k, which concludes the B_k if this event was the departure of a served request, with probability $p_k^{(dep)}$. In case of an arrival, in addition to the previously calculated area kI_k, the random variable recursively consists of the sum of B_{k+1} and B_k , because the number of jobs in the system has increased by one, and eventually returns back to k. Due to the memoryless property of the M/M/c system, the random variables are independent so they can be summed. Thus, the value of the B_k adds up to the total area under the time–number of busy pods chart, during the time interval T_k as illustrated by Figure A1. In the case of $k = c$, the area under the chart is the length of the interval T_c (see Definition A1) times the maximal number of busy servers c.

The distribution of the inter-event time I_k is exponentially distributed with rate $\lambda + k\mu$. The probabilities of an arrival or departure event if there are k busy pods in the defined aggregate Poisson process can be calculated [33]:

$$p_k^{(dep)} = \frac{k\mu}{\lambda + k\mu}; \quad p_k^{(arr)} = \frac{\lambda}{\lambda + k\mu} \tag{A26}$$

Analogously to Definition A2, we formally define the total weighted busy time Ω_c. Due to the memoryless property of the M/M/c system, the subsequent B_k values are independent random variables, so they can be simply summed.

Definition A4 (Weighted total busy time Ω_c of M/M/c)**.** *The weighted total busy time measures the area under the time–busy pod count function for an M/M/c system starting from c busy servers until the first idle moment:*

$$\Omega_c = \sum_{k=1}^{c} B_k$$

Both considered composite random variables Θ_c and Ω_c are composed of independent random variables as defined earlier, so their expected values can be calculated summing the expected values of the components:

$$\mathbb{E}[\Theta_c] = \sum_{k=1}^{c} \mathbb{E}[T_k] \tag{A27}$$

$$\mathbb{E}[\Omega_c] = \sum_{k=1}^{c} \mathbb{E}[B_k] \tag{A28}$$

In the following we show how to calculate these expected values using Laplace–Stieltjes Transform (LST).

Lemma A4 (Laplace–Stieltjes Transform (LST) properties [32])**.** *Let $F_X(x)$ be the CDF of X arbitrary random variable with LST $\varphi_X(s)$.*

$$aX \to F_X(\frac{x}{a}), \qquad \textit{where } a \in \mathbb{R}^+, \tag{A29}$$

$$F_X(\frac{x}{a}) \xrightarrow{LST} \varphi_X(as), \qquad \textit{where } a \in \mathbb{R}^+, \tag{A30}$$

$$F_{X+Y}(x) \xrightarrow{LST} \varphi_X(s)\varphi_Y(s), \qquad \textit{where } X \textit{ and } Y \textit{ are independent random variables}, \tag{A31}$$

$$\mathbb{E}[X] = -\frac{d}{ds}\varphi_X(0) \qquad \textit{is the expected value of } X. \tag{A32}$$

Referring to earlier works on the M/M/c analysis, the previously defined distributions' LSTs have been calculated [33], we state them here as lemmas.

Lemma A5 (Laplace–Stieltjes Transform of T_k [33])**.** *Let $\eta_k(s)$ be the Laplace–Stieltjes Transform of the cumulative distribution function (CDF) of busy period T_k:*

$$\eta_k(s) = \begin{cases} \frac{k\mu}{s-\lambda-\lambda\eta_{k+1}(s)+k\mu} & \textit{if} \quad 1 \le k < c, \\ \frac{c\mu}{s+\lambda-\lambda\eta_c(s)+c\mu} & \textit{if} \quad k = c \end{cases}$$

Using (A32) of Lemma A4 we can calculate the elements of the summation in (A27), which concludes the calculation of the first expected value:

$$c\mathbb{E}[\Theta_c] = c\sum_{k=1}^{c} \mathbb{E}[T_k] = -c\sum_{k=1}^{c} \frac{d}{ds}\eta_k(0) \tag{A33}$$

Lemma A6 (Laplace–Stieltjes Transform of I_k [33])**.** *Let the LST of the CDF of inter-event time I_k be $v_k(s)$:*

$$v_k(s) = \frac{\lambda + k\mu}{s + \lambda + k\mu}$$

Lemma A7 (Laplace–Stieltjes Transform of B_k)**.** *Let $\beta_k(s)$ be the LST of the CDF of weighted single busy period B_k:*

$$\beta_k(s) = \begin{cases} \frac{p_k^{(dep)} v_k(k\cdot s)}{1-p_k^{(arr)} v_k(k\cdot s)\beta_{k+1}(s)} & \textit{if}\, 1 \le k < c, \\ \eta_c(c \cdot s) & \textit{if}\, k = c. \end{cases}$$

Proof of Lemma A7. As B_k is recursively defined in Definition A3, we express its LST in a recursive form. Using the first two properties of Lemma A4, the LSTs of the random variables are scaled. Summation of the independent variables means multiplication in the LST-space as shown in Lemma A4. Using Lemma A6 and Lemma A5, the LSTs of inter-arrival time I_k and single busy period T_k are known. Applying the mixture distribution of random variables with probabilities $p_k^{(arr)}$ and $p_k^{(dep)}$ to the case $1 \leq k < c$ of Definition A3 we get:

$$\beta_k(s) = p_k^{(dep)} v_k(k \cdot s) + p_k^{(arr)} v_k(k \cdot s)\beta_{k+1}(s)\beta_k(s) \qquad \text{if } 1 \leq k < c \tag{A34}$$

Finally, arranging the equation to $\beta_k(s)$, and expressing $\beta_k(s)$ for the case of $k = c$, we get the statement of the lemma. □

Using (A32) of Lemma A4 we can calculate the elements of the summation in (A28), which concludes the calculation of the second expected value:

$$\mathbb{E}[\Omega_c] = \sum_{k=1}^{c} \mathbb{E}[B_k] = -\sum_{k=1}^{c} \frac{d}{ds}\beta_k(0) \tag{A35}$$

Dividing (A35) by the previously calculated expression (A33), we get the directly calculable fraction for the scaling metric estimation, as used in Proposition 2:

$$\tilde{u} = \frac{\mathbb{E}[\Omega_c]}{c\mathbb{E}[\Theta_c]} = \frac{\sum_{k=1}^{c} \frac{d}{ds}\beta_k(0)}{c\sum_{k=1}^{c} \frac{d}{ds}\eta_k(0)} \tag{A36}$$

□

References

1. Varga, P.; Peto, J.; Franko, A.; Balla, D.; Haja, D.; Janky, F.; Soos, G.; Ficzere, D.; Maliosz, M.; Toka, L. 5G support for Industrial IoT Applications—Challenges, Solutions, and Research gaps. *Sensors* **2020**, *20*, 828. [CrossRef] [PubMed]
2. Blanco, B.; Fajardo, J.O.; Giannoulakis, I.; Kafetzakis, E.; Peng, S.; Pérez-Romero, J.; Trajkovska, I.; Khodashenas, P.S.; Goratti, L.; Paolino, M.; et al. Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN. *Comput. Stand. Interfaces* **2017**, *54*, 216–228. [CrossRef]
3. Rost, M.; Schmid, S. Charting the Complexity Landscape of Virtual Network Embeddings. In Proceedings of the IFIP Networking Conference (IFIP Networking) and Workshops, Zurich, Switzerland, 14–16 May 2018.
4. Rost, M.; Schmid, S. Virtual Network Embedding Approximations: Leveraging Randomized Rounding. *IEEE/ACM Trans. Netw.* **2019**, *27*, 2071–2084. [CrossRef]
5. Laghrissi, A.; Taleb, T. A Survey on the Placement of Virtual Resources and Virtual Network Functions. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1409–1434. [CrossRef]
6. Kubernetes Documentation. Available online: https://kubernetes.io/docs/home/ (accessed on 27 March 2020).
7. Horizontal Pod Autoscaler—Kubernetes. Available online: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (accessed on 24 March 2020).
8. Fernandez, J.M.; Vidal, I.; Valera, F. Enabling the Orchestration of IoT Slices through Edge and Cloud Microservice Platforms. *Sensors* **2019**, *19*, 2980. [CrossRef] [PubMed]
9. Knight, S.; Nguyen, H.; Falkner, N.; Bowden, R.; Roughan, M. The Internet Topology Zoo. *IEEE J. Sel. Areas Commun.* **2011**, *29*, 1765–1775. [CrossRef]
10. Chowdhury, M.; Samuel, F.; Boutaba, R. PolyViNE: Policy-Based Virtual Network Embedding across Multiple Domains. In Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures—VISA '10, New Delhi, India, 3 September 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 49–56. [CrossRef]

11. Kumar, A.; Rastogi, R.; Silberschatz, A.; Yener, B. Algorithms for provisioning virtual private networks in the hose model. *IEEE/ACM Trans. Netw.* **2002**, *10*, 565–578. [CrossRef]
12. Ravindran, R.; Smith, P.A.; Wang, G.Q.; Zhang, H. Method and Apparatus for Computing Metric Information for Abstracted Network Links. US Patent 7,382,738, 3 June 2008.
13. Tran, A.K.; Piran, M.J.; Pham, C. SDN Controller Placement in IoT Networks: An Optimized Submodularity-Based Approach. *Sensors* **2019**, *19*, 5474. [CrossRef] [PubMed]
14. Karakus, M.; Durresi, A. A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN). *Comput. Netw.* **2017**, *112*, 279–293. [CrossRef]
15. Kang, N.; Liu, Z.; Rexford, J.; Walker, D. Optimizing the "One Big Switch" Abstraction in Software-Defined Networks. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies—CoNEXT '13, Santa Barbara, CA, USA, 9–12 December 2013; Association for Computing Machinery: New York, NY, USA, 2013; pp. 13–24. [CrossRef]
16. Gil Herrera, J.; Botero, J.F. Resource Allocation in NFV: A Comprehensive Survey. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 518–532. [CrossRef]
17. Sonkoly, B.; Szabo, R.; Jocha, D.; Czentye, J.; Kind, M.; Westphal, F.J. UNIFYing Cloud and Carrier Network Resources: An Architectural View. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; pp. 1–7. [CrossRef]
18. Vaishnavi, I.; Czentye, J.; Gharbaoui, M.; Giuliani, G.; Haja, D.; Harmatos, J.; Jocha, D.; Kim, J.; Martini, B.; MeMn, J.; et al. Realizing services and slices across multiple operator domains. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–7. [CrossRef]
19. Sonkoly, B.; Szabó, R.; Németh, B.; Czentye, J.; Haja, D.; Szalay, M.; Dóka, J.; Gerő, B.P.; Jocha, D.; Toka, L. 5G Applications from Vision to Reality: Multi-Operator Orchestration. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1401–1416. [CrossRef]
20. Fischer, A.; Botero, J.F.; Beck, M.T.; de Meer, H.; Hesselbach, X. Virtual Network Embedding: A Survey. *IEEE Commun. Surv. Tutor.* **2013**, *15*, 1888–1906. [CrossRef]
21. Xie, Y.; Liu, Z.; Wang, S.; Wang, Y. Service Function Chaining Resource Allocation: A Survey. *arXiv* **2016**, arXiv:1608.00095.
22. Moens, H.; Turck, F.D. VNF-P: A model for efficient placement of virtualized network functions. In Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop, Rio de Janeiro, Brazil, 17–21 November 2014; pp. 418–423.
23. Qu, L.; Assi, C.; Shaban, K.; Khabbaz, M.J. A Reliability-Aware Network Service Chain Provisioning With Delay Guarantees in NFV-Enabled Enterprise Datacenter Networks. *IEEE Trans. Netw. Serv. Manag.* **2017**, *14*, 554–568. [CrossRef]
24. Gouareb, R.; Friderikos, V.; Aghvami, A. Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2346–2357. [CrossRef]
25. Leivadeas, A.; Kesidis, G.; Ibnkahla, M.; Lambadaris, I. VNF Placement Optimization at the Edge and Cloud. *Future Internet* **2019**, *11*, 69. [CrossRef]
26. Németh, B.; Sonkoly, B.; Rost, M.; Schmid, S. Efficient service graph embedding: A practical approach. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 19–25. [CrossRef]
27. Sonkoly, B.; Szabó, M.; Németh, B.; Majdán, A.; Pongrácz, G.; Toka, L. FERO: Fast and Efficient Resource Orchestrator for a Data Plane Built on Docker and DPDK. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 243–251. [CrossRef]
28. Németh, B.; Szalay, M.; Dóka, J.; Rost, M.; Schmid, S.; Toka, L.; Sonkoly, B. Fast and efficient network service embedding method with adaptive offloading to the edge. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Honolulu, HI, USA, 15–19 April 2018; pp. 178–183. [CrossRef]
29. Holzmüller, D. Improved Approximation Schemes for the Restricted Shortest Path Problem. *arXiv* **2017**, arXiv:1711.00284.
30. Abbasi, A.A.; Younis, M. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.* **2007**, *30*, 2826–2841. [CrossRef]
31. Schaeffer, S.E. Graph clustering. *Comput. Sci. Rev.* **2007**, *1*, 27–64. [CrossRef]

32. Lakatos, L.; Szeidl, L.; Telek, M. *Introduction to Queueing Systems with Telecommunication Applications*; Springer Publishing Company, Incorporated: New York, NY, USA, 2013.
33. Omahen, K.; Marathe, V.M. Analysis and Applications of the Delay Cycle for the M/M/c Queueing System. *J. ACM (JACM)* **1978**, *25*, 283–303. [CrossRef]