Computer Communications 139 (2019) 103–122

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

The P-ART framework for placement of virtual network services in a multi-cloud environment

Lav Gupta [a,*], Raj Jain [a], Aiman Erbad [b], Deval Bhamare [c]

[a] Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, USA
[b] Department of Computer Science and Engineering, Qatar University, Doha, Qatar
[c] Department of Mathematics and Computer Science, Karlstad University, Sweden

ARTICLE INFO

Keywords:
Virtual network services
Network function virtualization
Service function chain
Virtual network function
Multi-cloud systems
Machine learning
Dynamic placement

ABSTRACT

Carriers' network services are distributed, dynamic, and investment intensive. Deploying them as virtual network services (VNS) brings the promise of low-cost agile deployments, which reduce time to market new services. If these virtual services are hosted dynamically over multiple clouds, greater flexibility in optimizing performance and cost can be achieved. On the flip side, when orchestrated over multiple clouds, the stringent performance norms for carrier services become difficult to meet, necessitating novel and innovative placement strategies. In selecting the appropriate combination of clouds for placement, it is important to look ahead and visualize the environment that will exist at the time a virtual network service is actually activated. This serves multiple purposes — clouds can be selected to optimize the cost, the chosen performance parameters can be kept within the defined limits, and the speed of placement can be increased. In this paper, we propose the P-ART (Predictive-Adaptive Real Time) framework that relies on predictive-deductive features to achieve these objectives. With so much riding on predictions, we include in our framework a novel concept-drift compensation technique to make the predictions closer to reality by taking care of long-term traffic variations. At the same time, near real-time update of the prediction models takes care of sudden short-term variations. These predictions are then used by a new randomized placement heuristic that carries out a fast cloud selection using a least-cost latency-constrained policy. An empirical analysis carried out using datasets from a queuing-theoretic model and also through implementation on CloudLab, proves the effectiveness of the P-ART framework. The placement system works fast, placing thousands of functions in a sub-minute time frame with a high acceptance ratio, making it suitable for dynamic placement. We expect the framework to be an important step in making the deployment of carrier-grade VNS on multi-cloud systems, using network function virtualization (NFV), a reality.

1. Introduction — challenges and contributions

Carriers perceive Network Function Virtualization (NFV) as a disruptive technological development that has the potential of delivering them from the problems of the traditional physical networks. NFV allows network functions and appliances to be instantiated in software on computing and networking resources obtained from datacenters or cloud service providers. The concoction of NFV and cloud computing holds a great promise for carriers. It promises to deliver freedom from vendor dependence and expensive proprietary equipment, ease of service creation and phasing out, the flexibility of scaling and descaling, having points of presence closer to the users and avoiding a single point of failure. Cloud computing and Network Function Virtualization have a natural synergy that awaits full exploitation. It is expected that these two powerful paradigms would evolve together to support the requirements of virtual network services (VNS). The European Telecommunications Standards Institute (ETSI) specification of classification of cloud-native VNF implementations describes the creation of VNFs on different types of clouds [1].

One of the biggest challenges in deploying NFV over multiple clouds today is the low VNS performance. There is a general concern regarding the current technological capability to extract carrier-grade performance from NFV-based services [2,3]. The Internet Engineering Task Force (IETF) has also identified performance and guaranteeing the quality of service as open research areas and technology gaps in NFV [4]. The performance standards have been strict in telecommunications networks, with International Telecommunications Union (ITU) standards being adopted by most administrations. The standards prescribe stringent control over performance parameters like latency, jitter and packet loss [5]. The availability requirement is of the order of five nines (permissible downtime of just 26 s in 30 days).

* Correspondence to: Department of CSE, Washington University in St Louis, St Louis, MO 63130, USA.
E-mail addresses: lavgupta@wustl.edu (L. Gupta), jain@wustl.edu (R. Jain), aerbad@qu.edu.qa (A. Erbad), deval.bhamare@kau.se (D. Bhamare).

https://doi.org/10.1016/j.comcom.2019.03.003
Received 26 October 2018; Received in revised form 20 January 2019; Accepted 1 March 2019
Available online 18 March 2019

There are a number of reasons why the software versions of the network functions, i.e., Virtual Network Functions (VNFs), do not give a performance that is comparable to the purpose-built physical appliances used in the traditional networks. As anyone would guess, the main reason is the inability of the network functions created in software over general-purpose hardware, in matching the performance of specialized hardware-based functions. The performance suffers further when these 'softwarized' functions are instantiated over clouds. To compound the problem, carriers have lesser control when network appliances move from their own switch rooms and transmission centers onto the Cloud Service Providers' (CSPs') virtual machines (VMs). Add to this the newfound ease of creation, destruction, migration, and scaling of virtual resources (courtesy NFV), and opportunities for indiscriminate virtualization proliferate. All of these issues cause performance to go downhill. Previous work has shown that virtualization may lead to abnormal latency variations and significant throughput instability [6]. In their infrastructure overview, ETSI has indicated latency and throughput constraints as the discouraging factors for the use of public clouds for hosting NFV. Even though researchers have proposed ways of improving the performance of virtual network functions [7,8], legitimate concerns still remain. All said and done, the advantages of the VNSs are far too important for researchers in academia and industry to forge ahead.

In the VNS game, carriers and CSPs may not always have a cordial relationship. It is challenging to co-optimize their conflicting goals when they collaborate to provide VNSs. Carriers look for standards-grade performance and availability at the minimum cost and in the desired time frame. So, not to take any chances, they incorporate these in their Service Level Agreements (SLAs) with the CSP. On the other hand, the CSPs aim to maximize the utilization of their physical and virtual resources to improve their profit margin.

In this paper, we make a case for the P-ART framework that will help CSPs alleviate some of the main concerns of carriers while deploying services — meeting the contracted performance and keeping the cost within the prescribed budget. The main contributions of this paper are summarized below:

1. We develop techniques for improving the performance of deployed VNSs through the following:

   (i) We propose an innovative predictive dynamic placement algorithm that takes care of changes in the state of the cloud environment to ensure the validity of the placement at the time of activation of a service. In addition, we propose placing complete chains rather than the commonly followed path of placing VNFs individually, to yield better results. As most carrier services are affected by latency, we choose to work with latency as an important performance measure. The work can be extended to other parameters following the same guiding principles.
   (ii) Since a public dataset suitable for the problem is scarce, we generated realistic datasets to train and test the models. To be doubly sure, we used a dataset obtained by building a queuing-theoretic model and another by implementing the system on CloudLab [9].
   (iii) One of the important parts of the framework is a novel method that refines the prediction algorithm by taking into account variations in network latency because of temporally varying traffic conditions in the carriers' networks. Unattended, such variations cause a concept-drift, which makes predictions unreliable and affects the accuracy of predictions. For this, we introduce a novel concept of using time as a feature in training the predictive machine learning models. The resulting use of multiple models makes the framework adaptive to diurnal traffic variations.
   (iv) Short-term traffic changes, because of events like a football match or an election rally, do not follow a pattern like diurnal traffic variations and need a different way of handling. Since retraining of models is a time consuming and expensive operation, the framework uses incremental learning to keep the models up-to-date.

2. We propose multiple criteria optimization through an innovative placement strategy. Specifically, placements are carried out to optimize cost and keep latency within the specified threshold. We explain in the related works section that, in general, ILP and its variants give optimal solutions but take significantly more time than other methods. This limits their utility in responding fast to the change of state of the multi-cloud system and the subscriber demands from the service during its actual operation. To the best of knowledge, the random optimization as a viable method to achieve optimized placement has not been used before. The algorithm converges to the global minimum even in the case of a multi-modal dataset.
3. We incorporate in our framework, innovative techniques for making the placement fast with high acceptance rate. The high speed of placements allows the CSP to make changes in the network dynamically, in real-time or near real-time, as the factors like demand, traffic congestion on links, availability of resources on various clouds change. A high acceptance rate implies that a placement attempt would be successful every time if enough resources are available on the clouds.
4. Finally, the ideas explained above are brought together to form the P-ART framework for dynamic predictive, adaptive and real-time placement of carrier virtual network services.

In the preliminary version of the paper, presented at an IEEE conference in 2017, the contributions mentioned in 1(i), 1(ii), 2 and 3 were explored [10]. The new work explained in 1(iii), 1(iv) and 4 enables us to report the complete framework in this paper. The rest of the paper is organized as follows. In Section 2, we discuss the VNS environment. This section also serves to clarify the terminology used. Section 3 presents a summary of the related work and how this work is different from other previously reported solutions. The problem description is in Section 4. The P-ART framework is discussed in Section 5. In Section 6, we present the evaluation results. Finally, Section 7 gives a summary and describes the ongoing work.

2. Virtual network service environment

The network services are voice and data services, wired or wireless, provided by telecommunication companies (referred to as carriers in this paper). These network services include public services like mobile telephony, broadband and Internet, content delivery, enterprise networks, leased circuits, and virtual private networks. Traditionally, networks providing these services have been built using physical appliances and transmission links that are custom built for carrier-grade performance. This physicality usually creates vendor lock-in, prolonged service deployment time, inflexibility in scaling and introducing new services, and high cost. NFV and cloud computing provide a way to create network functions, in software, over inexpensive virtual resources. Such virtual functions can be linked with virtual network resources to create VNSs. The VNSs result in flexible, scalable and less expensive networks that are not proprietary and prevent vendor lock-in. We shall see the constituents of VNS in this section along with the cloud set-up that can be used for hosting such services.

2.1. Constituents of a virtual network service

In most discussions on VNSs, VNFs are the basic unit of placement. VNFs are software-based implementations of physical network functions that are used in traditional carrier and enterprise networks. They exhibit functional behavior similar to their physical counterparts and have well-defined interfaces consistent with relevant industry standards. VNFs can be instantiated on virtual machines (VMs) obtained from datacenters, or from cloud service providers. All the instances of a VNF, say the core router function, would usually be hosted on one or more dedicated VMs on one or more clouds depending on the carriers' requirements and CSPs own policies regarding these deployments.

A Service Function Chain (SFC) or a VNF forwarding graph is a set of VNFs interconnected to route the packets in a well-defined

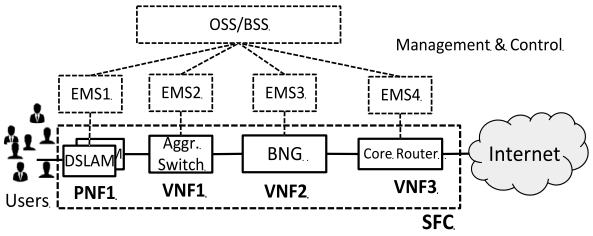


Fig. 1. Broadband service function chain and associated modules.

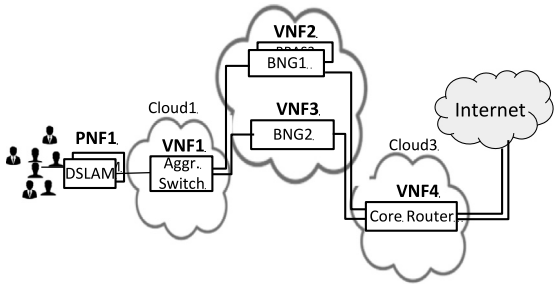


Fig. 2. Mapping service function chain to the multi-cloud system.

sequence [11]. They are connected like the physical appliances are connected in a traditional network [12]. IETF RFC 7498 [13] describes each network service (NS) being implemented through one or more service function chains (SFC) [14]. The carrier may like to retain some of the legacy physical network functions (PNFs) while virtualizing the other functions. The SFC may, therefore, consist of VNFs, PNFs, and links among them. Fig. 1 shows the components of an SFC and associated modules.

The broadband VNS, shown in Fig. 1, is an SFC consisting of four VNFs, viz., an aggregation switch, two types of Border Network Gateways (BNGs) and a core router. It also has multiple instances of a Physical Network Function (PNF), viz., Digital Subscriber Line Access Multiplexers (DSLAMs), retained from the legacy network. Each VNF has its own Element Management System (EMS), which interfaces the VNF to rest of the network [12]. The Operation Support System/Business Support System (OSS/BSS) of the carrier manages the VNFs and SFC through the EMSs.

SFCs can be placed on the available clouds in a number of ways. CSPs may offer commonly used network functions in the form of VNF-as-a-Service (VNFaaS), which may be a part of an SFC. Alternatively, a carrier may lease virtual resources in the clouds and instantiate VNFs itself, with a view to exercise more control over performance parameters and cost. Our discussions presume the use of the latter method. Fig. 2 shows an example of an SFC mapped to multiple clouds. It may be noted that we now have four VNFs as the SFC has two types of BNGs. The Aggregation Switch is presumed to have a built-in load-balancing function for distributing traffic between the two forked paths. The end-to-end latency of the service function chain would depend on how, when, and where the constituent functions have been placed. The users shown in the figure are customers of the carrier while the carrier is a tenant on the cloud system. When the initially placed SFC does not meet the required conditions, operations, like moving around the VNFs in the clouds or scaling up the number of instances, would be resorted to.

2.2. The multi-cloud hierarchy

There are public cloud services like Amazon EC2, Google Cloud Services, and Microsoft Azure that provide the advantage of a relatively inexpensive resource leasing solution. Big public clouds are multi-tenant and have a regional or international presence. These clouds can handle large volume, variety, and velocity of traffic. Large public clouds do offer greater flexibility in obtaining resources and more analytical sophistication, but taking all the data to just one public cloud would create traffic congestion and increase the access latency. Using a single cloud could often result in a single point of failure in the case of cloud blackouts, which are not uncommon.

Additionally, the points of presence (PoPs) of large public clouds may not be close to the subscriber clusters and may give rise to increased access latency. If the application calls for lower access latencies then edge clouds may offer a good solution. Carriers may also have their own private clouds, which they can customize and exercise more control over. This hierarchy of clouds – mobile-edge, private, and public – forms a multi-cloud system to provide a combination of features like low latency, high storage, complex computations, lower cost, and better security.

2.3. Representation of the tenant profile

In this work, a cloud tenant (in our case, a carrier) profile is represented as a tuple $\langle c_N, v_1, v_2, \ldots, v_m, p\rangle$ for each request. Here, $v_1, \ldots, v_m$ represent the VNFs and the order of traffic traversal in a linear chain. The term c_N is the native cloud for the tenant to which it is parented and through which the traffic enters an SFC and p is the desired packet rate (packet/second). Multiple tuples can be used to represent branched traffic flows. Other stipulations like latency threshold (L_{th}) are part of the SLA. All the requests of the tenant are consolidated to calculate the required number of instances of each VNF and inter-VNF links of appropriate capacities. The cloud topology may be represented by the graph G_c = (C, T), where C is the set of available clouds $\{c_1, c_2, \ldots, c_k\}$ and $t_{i,j}$ are the inter-cloud links. The CSP (or a cloud broker who integrates services from multiple clouds) carries out the task of mapping service chains onto the available clouds to achieve optimal results for the carrier. In our case, optimality refers to the least-cost solution that meets the end-to-end latency threshold requirement.

3. Problem definition

In this section, we summarize some of the key outstanding problems in the dynamic placement of carrier VNSs, in a multi-cloud environment that we attempt to handle in the P-ART framework described in this paper.

3.1. Achieving dynamic placement in multi-cloud systems

Some carrier services may be fairly static, e.g., fixed voice network. Thus, over time the number of instances of VNFs and link capacities required only change slowly over time. On the other hand, some services may be extremely dynamic, requiring a change in number and types of VNF instances, re-dimensioning of links and changes in the offered features of the service very frequently. An example of such a service would be an intelligent network service like televoting in a TV reality show. Different TV reality shows may require different features and the number of voters may swing unpredictably during the voting window. If the CSP only offers largely static placement with reactive and relatively slow modifications, then the carrier's requirements may not be met.

The bottom line is that both, the dynamic and static services would require the CSP to scale VNF capacities or links, albeit at a different rate. However, dynamic services may be more demanding in terms of types and number of instances of VNFs and link resources and may even require migration of VNFs from one cloud to another to be able to continuously meet the cost and end-to-end latency constraints. A dynamic placement algorithm, that monitors the SLA parameters and proactively causes changes in the amount of resources and the combination of clouds to meet all the requirements, is still a challenging issue.

3.2. Optimizing the SFC performance

When the data are high dimensional and multi-modal, optimizing placement of individual VNFs may not achieve the global minimum. Placing SFCs as a unit yields better results. The opportunity to achieve the global minimum for the parameter being optimized is available when placing the SFC. If sufficient resources are not available to implement full-service chains, then the request may be rejected or, if the policy permits, degraded service (for instance without firewall) is provided [11,15]. In this paper, we only consider complete SFC placement. The case where the customer accepts degraded performance due to low-capacity chain placement or partial functionality due to incomplete chain placement would be taken up in future work.

3.3. Meeting the cost and latency constraints

From the carrier's perspective, the placement problem boils down to placing network functions to meet the cost and latency objectives. At the commencement of the VNS and during operation, the placement problem needs to be repeatedly solved to ensure that the carrier requirements are continually met. Performance criteria vary from service to service. For the carrier services like voice, broadband, and content delivery some of the common factors are jitter, packet loss, latency, and throughput. ITU standards for QoS parameters in carrier networks are available in [5]. Latency is one of the most important criteria, and we have taken that as a reference performance parameter. The framework can be extended to include other criteria as well.

3.4. Speed and accuracy of the placement

Carriers want short placement and reconfiguration time so that the solution can be useful in an operational network. The CSP wants the solution to have the high success of placement requests such that utilization of the virtual resources increases. When the system cannot place despite the availability of resources, CSPs lose by way of unused resources and possible breach of SLA.

3.5. Interference among VNFs

The CSP may instantiate a number of VMs on a physical machine (PM) and a number of virtual links on the physical inter- and intra-cloud links. VNFs of more than one service provider may be instantiated on the same PM. In some cases, pre-instantiated VNFs may be shared among carriers. Sharing of virtual resources does not only cause performance concerns but could also give rise to security concerns. In this paper, we have presumed that VNFs of different types belonging to a carrier are on different VMs.

3.6. Problems addressed and not-addressed in this paper

The following issues have been specifically addressed in the paper:

(a) Dynamic placement of the complete SFCs belonging to a VNS.
(b) Meeting the specified performance and cost criteria.
(c) Prediction of latency using machine learning as a basic input for the placement algorithm.
(d) Refining the prediction by handling the temporal variation of traffic, unplanned short-term spikes in traffic and the time lag between planning and commissioning of SFCs.
(e) A fast placement algorithm that places with high success rate.

The following problems are left for future work:

(a) Use of under-dimensioned service chains
(b) Security issues of the VNSs.

4. Related work and how this research advances the state-of-the-art

A review of recent publications shows a strong interest of researchers in the problem of placement in the context of NFV. We discuss here some of the relevant works published during the last two years to show how the field has progressed. There is some older useful research on which many of the recent works build, and these have been cited in the works that have been examined. Since our research is in the area of cost and latency optimization, we focus on research dealing directly (for example by optimizing cost or latency) or indirectly (by optimizing utilization of resources thereby reducing cost) with these aspects. We conclude this section by elaborating how our work advances the state-of-the-art.

4.1. Review of recent works on VNF placement

4.1.1. Methods based on ILP and its variants for optimization

In [16] the authors contend that unlike most other works they have considered QoS/SLA along with resource requirement of network services. They show that the virtualization overhead increases with traffic load and the number of VMs due to factors such as scheduling delays, context switching, and flow routing. The authors include virtualization overhead while setting up their MILP model to optimize resource usage while guaranteeing latency requirements. The model optimizes the cost including the utilized processor, memory and physical links under the latency constraint of maximum round-trip time. It is seen that for a network with 28 nodes and 41 links the model takes about an hour to arrive at an optimum solution. The authors in [17], use an MILP model to optimize network latency and increase the acceptance rate of strict delay requirements. One of the constraining factors in evaluation is the location of all the VNFs in the same cloud. It is also somewhat unclear how the method will scale from 5 VNF to a large network, for delays. The algorithm chooses a more expensive path to ensure a minimum delay. An intuition that probably does not require proof is that delay will be more with high bandwidth requirement, or when more requests seek the same link. In cases where the number of requests is high, the solver is not able to find an optimal solution in the joint delay and routing cost optimization problem. The solution for the optimal chaining and routing with MILP limits the scale of the problem.

4.1.2. ILP and heuristic to speed up ILP

In [18], the authors optimize the number of physical machines (PM) used using an ILP model. They take into account the time-varying workloads while instantiating VNFs in PM. A two-stage heuristics solution has been suggested to solve the ILP, with a correlation-based greedy algorithm as the first stage and a further adjustment at the VNF in each SFC as the second. The simulation demonstrates improved utilization of network resources and reduced number of PMs compared to the benchmarks. This and some other works presume multi-tenant VNFs to improve utilization. While this may be good from the point of view of cloud service providers, but carriers would usually request exclusive VNFs hosted on exclusive VMs because of security and performance concerns. In [19] the authors propose placement of VNFs in the edge clouds to minimize end-to-end latency. Using and ILP model, the authors show that cloud-only deployments gave more than 3 times more latency than cloud-and-edge deployments. The absolute times for initial placement and for each re-configuration are not known. They also present a way to dynamically re-schedule the optimal placement of VNFs based on temporal network-wide latency fluctuations using optimal stopping theory. Scheduling re-optimization may reduce latency violations, but they may require an increased number of migrations. Periodic migration also has a problem, as it requires human intervention to decide on the periodicity of tuning. The authors suggest a method using optimal stopping theory to select the right time for placement.

4.1.3. ILP and heuristics for comparison

In [20], the authors consider an IoT-edge cloud–main cloud scenario in a dynamic multi-user situation. The authors set up an MILP model to minimize the end-to-end communication delay while keeping the cost to the minimum. However, they realize that the MIP formulations rapidly increase in complexity and take a long time to give an optimum solution, as the problem becomes large. To counter this, the authors also propose Tabu search for placement and chaining. They find that the MIP method takes 200 times slower than the Tabu Search. The authors in [21] solve VNF placement and chaining problem as ILP and also propose another method called Cost-efficient Centrality-based VNF Placement and chaining algorithm (CCVP). The objective is to minimize the cost by finding an optimal number of VNF instances and their locations for handling the required traffic. To simplify they assume that the network provider is the owner of NFVI so concerned factors are under its control. The CCVP is based on the Betweenness centrality algorithm. The high centrality indicates that a vertex of a graph G can reach other vertices on relatively short paths. This results in lower network cost. They show that the overall cost of their method is close to ILP. It should be noted that processing delays and link bandwidths are not considered in the analysis. In [22], the authors pursue the objective of optimization of energy consumption as an ILP model. This purportedly gives a reduction in the operational cost of the placement. They also propose a near-optimal approximated algorithm to solve the problem using the Markov approximation technique. They show that their algorithm can achieve the performance arbitrarily close to the global optimum. Simulation results show that the algorithm saves up to 14.84% energy consumption compared with previous VNF placement algorithms.

4.1.4. Non-ILP heuristic solutions

In [23] the authors presume sharing of VNFs among different service chains. It should be noted that while sharing may improve VM utilization, it might consume more link bandwidth because these chains may need to go through a longer path in order to reach the shared VM. As mentioned before, from carriers' point of view this arrangement may give rise to security issues as well as make it difficult to control latency. The authors contend that most of the existing works are mainly targeted on improving VM utilization, without considering the required bandwidth resources. This paper has examined the joint VNF placement and Path Selection problem, so as to maximize the served traffic demands. In [24], the authors discuss a proactive placement model in the context of a content distribution network (CDN). They argue that VNF chaining and placement affect QoS, and formulate an optimization problem to find the optimal number of locations as well as efficient chaining such that the CDN cost is minimized and QoS is satisfied. The authors set up the problem as a bin-packing problem that involves selection of bins (surrogate servers) and dropping the items (VNFs) into them. The authors conclude that while their solution gives fewer servers but may give a high communications cost. In [25], the authors investigate the optimal placement of virtual resources to minimize the average response time in mobile edge computing (MEC) environment with a capacity constraint on the edge network. They use OEPA (Optimal Enumeration Placement Algorithm) as a benchmark to compare Latency-Aware Heuristic Placement Algorithm (LAHPA), which has lower computation complexity, Clustering Enhanced Heuristic Placement Algorithm (CEHPA) to enhance the performance of LAHPA, Substitution Enhanced Heuristic Placement (SEHPA). SEPHA turns out to be better than LAHPA. CEHPA and outperforms LAHPA and both are better than the general Greedy Placement Algorithm. The authors in [26] describe a dynamic placement algorithm based on traffic variations that saves operational expenditures. Their algorithm consolidates VNFs in the fewer possible number of network nodes while maintaining low blocking probability and guaranteeing latency targets to the supported services. They reuse VNFs, select VNFs based on locality and activate them based on the shortest path. The authors claim that their algorithm is able to balance the trade-off between minimizing latency violations, decreasing blocking probability and reducing operational expenditure. The success rate of the algorithm has not been mentioned. The authors claim 50% saving in telecom operators cost.

4.2. How does this work advance the state-of-the-art?

A carrier's environment is essentially different from an IT application environment. Carriers assiduously follow norms that have long been enforced by standardization agencies like ITU or through self-imposed discipline. They are generally loath to give these good practices up, even if that would mean marginally sacrificing on other competing cost objectives. Some of these practices relate to five nines reliability, guarding against inadvertent or malicious interaction of services (for example, because of VNFs being on the same servers or VNFs sharing the same VM) and having well-defined points of interconnections. Another important aspect is ensuring the security of their services. Some of these may be required by regulation to account for revenue generation by different networks or to have non-contentious sharing among carriers in case of multi-domain services.

There are a number of important factors that go into the planning of carriers' network services. The locality of VNFs, for instance, those belonging to the access network (like Radio Access Network), should ensure that the VNFs serving a cluster of subscribers are instantiated close to them to reduce cost and latency. There are a number of virtual functions that have an affinity and need to be placed as close as possible. In a broadband network, the edge routers may be connected to two core routers in order to ensure that large clusters of subscribers are not cut off from the network. In such a case, the cost of connectivity would be exorbitant if edge routers are generally located far away from the core routers. In the case of carrier's VNSs deployed over clouds, it must be remembered that the cloud resources (or the NFV resources) may not all belong to the carrier. In such a case, when the placement solution deals with packing the VNFs into physical or virtual machines, it generally helps the cloud service providers to reduce their cost. The carrier's objectives of isolation of services, security, affinity and QoS parameters may be jeopardized.

Unlike most other papers that deal with placing VNFs on virtualized datacenter resources or single clouds, this paper presumes a multi-cloud environment. Rather than optimizing the utilization of physical or virtual resources, it assumes carriers' viewpoint and optimizes, under latency constraint, the total cost of placement of network functions, which includes resources on various clouds and links. The cost is presumed to be adjusted to contain the apportioned capital and operational costs for the virtual network service under deployment. The method that we propose falls in the category of dynamic and proactive placement algorithms rather than being either of those. Our objective and constraint-based determination of clouds, on which the SFC will be placed, removes the tight binding between resources and the VNFs of the SFC. During operation, the placement is frequently re-evaluated to ensure continued optimality. We avoid the ILP route and use machine learning for placement, which reduces the time taken even for large placements and renders the re-evaluation problem trivial. If required, new placement and virtual resource dimensioning will be done consistent with the carrier SLA requirements and CSP policies. Selection of clouds for placement of chains of VNFs is based on the prediction of the state of the clouds at the time of placement. A number of innovations have been proposed in this part of the work. One such refinement is the compensation of concept drift due to diurnal variation of traffic. The methods adopted also lead to the high efficiency of the placement process, which ensures that placement requests are successful in all cases where enough capacity is available and constraints can be met.

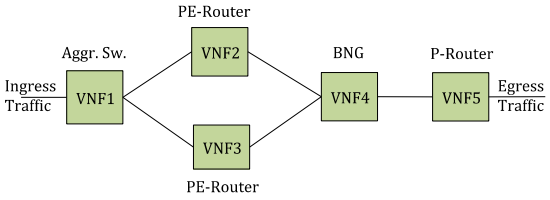


Fig. 3. The configuration of the experimental service chain.

5. The proposed P-ART framework

In this section, we describe our framework with approaches to solutions for the problems mentioned in Section 3 and for achieving the objectives specified. We also describe how the refinements mentioned were carried out to achieve the solution that can be used for carrier networks as well as in the enterprise environment. For our studies, we will consider the placement of the SFC shown in Fig. 3.

5.1. Information available from carriers and CSPs

Carriers, who request service chain placement, provide information about the performance requirement for a VNS, and the number and structure of SFCs and VNFs to be instantiated. A VNS may have one or more SFCs. The ith SFC S_i can be represented in terms of the constituent VNFs, i.e.,

$$S_i = \langle C_N, \mathrm{vnf}_1(i), \mathrm{vnf}_2(i), \ldots, \mathrm{vnf}_n(i), p \rangle \quad (1)$$

where C_N is the native cloud and p is the maximum packet rate through the chain. The native cloud is usually the point of presence (PoP) of the CSP closest to the carrier and provides interconnection to the carrier. The CSP may provide an option to connect at PoPs located at other places. This gives a choice to the carrier to have traffic ingress points close to the customers. The design is to be carried out such that the costs of the network, as well as latency in reaching the cloud system, are kept to the minimum or below a given threshold value.

An SFC is represented as a forwarding graph of the type G_v = (V, E), the nodes V being virtual network functions and edges E the virtual links among these functions. The demanded capacity of ith VNF, vnf_i ($i \leq n$) is expressed as v_i^c in the same integrated units as the cloud capacities (shown in Table 2). An integrated figure represents the compute capacity c_k, of a cloud k, consisting of a certain amount of processing, memory and storage components. However, there is no integer constraint on the VNF capacities. These are mapped onto resources in the available clouds represented as another graph G_c = (C, T), where C represents the set of clouds with physical/virtual infrastructure and T the set of links t_{ij} among them. The state of a cloud k at any time would involve the cloud compute and link capacities — installed capacities denoted as $c_k^{(c)}$ and $t_{kj}^{(c)}$, and the corresponding used capacities are $c_k^{(u)}$ and $t_{kj}^{(u)}$. The tenant carrier provides the maximum expected packet rate p for each request originating from a cluster of subscribers. The expected end-to-end latency is specified by the carrier in terms of a latency threshold (L_{th}). The CSP consolidates the VNF requests and packet rates required for each type of chain to allocate resources in an optimum way. Table 1 gives the symbols frequently used in the paper

Some of the important constraints subject to which the cost optimization is carried out are:

- The number of instances of each type of VNF across all the used clouds, for any carrier, should not exceed the number of licenses for that function type paid for by the carrier.
- To place any chain, at least one instance of each type of VNF needs to be instantiated.

Table 1
Symbols used.

Symbol	Description	Symbol	Description	Symbol	Description
c_k	Cloud k	c_N	Native cloud	$c_k^{(u)}$	Used capacity of cloud k
C	Set of all clouds available	$v_i^{(c)}$	Capacity demand for VNF i	$t_{ij}^{(u)}$	Used capacity of the link between clouds i & j
t_{kj}	Link from cloud k to j	$c_N^{(c)}$	Equipped cap of native cloud	p	The maximum expected packet rate
T	Set of all inter-cloud links	$c_N^{(u)}$	Used cap of native cloud	m	No of clouds selected
v_i	ith VNF	$c_k^{(c)}$	Installed capacity of cloud k	vnf_i	The ith VNF in the SFC
V	Set of VNFs	$t_{ij}^{(c)}$	Capacity of link between clouds i & j	L_{th}	Latency threshold
n	Types of VNFs	$V_i^{(c)}$	Capacity demand for ith VNF	C_B	Cost budget

- The total capacity of each type of VNF placed on any cloud k should not exceed the capacity available in the cloud.
- At any given time the sum of the traffic flows, due to all service chain placements, between any two clouds k and j should not exceed inter-cloud link capacity $t_{kj}^{(c)}$.
- The end-to-end latency, L, of any chain should not exceed the specified threshold L_{th}.
- While the cost is optimized, the carrier may additionally specify a budget C_B for it.

The framework requires that the CSP lays down its policies regarding tariffs, integrated virtual resource capacities, clouds offered, the arrangement with other cloud providers, cloud and link capacities offered, etc.

5.2. Predictive adaptive real time strategy

The proposed placement solution optimizes cost and constrains the end-to-end latency below the specified threshold, L_{th}. We assume that the design for instantiation of SFCs, belonging to a VNS, is ready at time t, but actual placement is yet to happen. In other words, the placement problem has been solved at time t for the placement and activation that will actually take place at time t_1. Predictive placement is used to take care of the change of state because of this time difference. Using prediction of the latency as the basis of design also takes care of the large number of infrastructure and network level parameters that interact in a complex way to decide the end-to-end latency. In addition to these, the background traffic in the network affects the latency experienced by the subscribers of the VNS being placed. Therefore, taking care of the diurnal traffic variations in the network makes the prediction of latencies more accurate and system more adaptive to such changes [27]. Short-term surges in traffic, due to events like a football match, would affect latency during the event and should be accommodated by dimensioning and reconfiguring the SFCs. This renders the system more responsive (and near real-time) in terms of latency predictions. We have taken into account all these factors in formalizing our prediction algorithm. Latencies so predicted are then used to select a suitable subset of least-cost clouds meeting the latency constraint. The complete algorithm is given in Algorithm 1.

Algorithm 1: PLACE_SERVICE_CHAIN (client_demands, csp_data, cv_model)

1: Set up cloud data // all $c_k \in C$ and $t_{k,j} \in T$
2: Set up client data // all $v_i \in V$
3: Latency threshold←L_{th}
4: Cost budget ← C_B
6: *NCloud* ← c_N // Native Cloud
7: v_i^c ← capacity demands for vnf_i
8: n ← length of the service function chain (number of VNFs)
9: *native* ← true // set native to 1 if native cloud is used else 0
10: if (*native* == 1) //place as many VNFs as possible in the native cloud
11: for v_i, i =1, n
12: if $c^c{}_N - c^u{}_N > v_i^c$ // native cloud has unused capacity
13: pop v_i
14: $c^u{}_N \leftarrow c^u{}_N + v_i^c$ // update cloud capacity
15: else
16: break
17: end if
18: end for
19: end if
20: if V != 0 // for remaining vnfs
21: call **RANDOM_SELECTION(*C*, *cv_model*, *r_clouds*)** //get a set of lowest cost clouds
22: sort ascending *r_clouds* on cost //set of smallest latency clouds
23: while V != 0
24: place vnfs //on sorted clouds
25: update capacity
26: update bandwidth
27: update vnfs_placed status
28 : end while
29: end if
30: if all_vnf_placed & latency of chain < L_{th} & cost of chain < C_B
31: output placement details
32: else
33: report failure to place
34: end if

The essential elements of the placement process can be understood like this: the placement process takes care of the change of state of the cloud system by predicting latencies at the time of actual activation of the SFCs. This obviates the need for drastic changes soon after placement or reconfiguration. Prediction is, thus, an essential element of the framework. Having said that, the prediction methodology needs to be robust against traffic variations. With this, the framework becomes adaptive to placement time and traffic variations. To make the framework fast, responsive, and useful in real-time, further steps need to be taken. For this, short-term traffic variations are taken into account. Two other important factors that need to be taken into account are *speed* and *acceptance rate* of placement. Fast placement algorithms would allow continuous optimization by making real-time changes (e.g., migration) possible when the need arises during the operation of the network. For dynamic scaling, a fast algorithm would be able to place hundreds or thousands of functions in sub-minute time frame. Concurrently, a 100% acceptance rate implies that the algorithm is able to satisfy all requests for placing SFC, subject to capacity being available. This contributes to the avoidance of repeated attempts and saves time.

Algorithm 1 is called for placement and reconfiguration. The cloud and client data are initialized based on the CSP resources and the client request and policies (*lines 1–5*). A separate process produces a trained model *cv_model* using the training data (**X** ← feature_set and **y** ← labels), which is available to the placement procedure. The placement normally begins with the native cloud (this can be overridden in *line 9* by setting *native* = 0). The algorithm accommodates as many VNFs as possible in the native cloud (*lines 10–18*). For the remaining VNFs, the SVR module predicts the latency of various clouds. This algorithm uses Algorithm 3 (procedure RANDOM_SELECTION) to select the set of

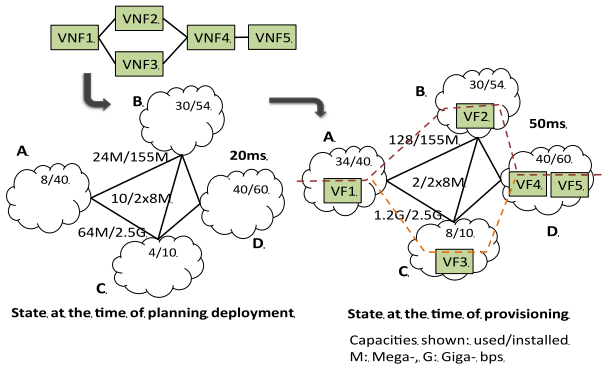


Fig. 4. Need for predictive placement.

m least-cost clouds that meet the latency requirements. The number *m* can be decided to start with enough capacity to place all the VNFs. For the least-cost set, the algorithm calculates the assignment of VNFs in the sequence in which they appear in the SFC. The final cost and latency are reported (line 31). If the clouds are exhausted, and placement has not completed, then failure to place is reported. If this case happens frequently, then the number *m* needs to be increased.

5.2.1. *Predictive placement for handling change of state of the system*

The cost of placing an SFC is a function of the set of clouds C_s ($C_s \subseteq C$), where C is the set of all available clouds), selected to place the virtual network functions and the amount of computing, storage, and networking resources consumed. End-to-end Latency (L) of the SFC depends on a number of factors prominent of which are, (a) the installed and used capacities of computing, networking and storage resources in the physical servers and the links, (b) the traffic pattern on the links, (c) the types of network functions sharing the servers, and (d) the distance between clouds. These factors together constitute the state S_t of the multi-cloud system at time t.

As the system operates, the number of tenants and their workloads change, the state also changes. The amount of latency introduced in a placement by the state of the cloud, therefore, changes over time. Given the state S_t, latency can be computed by using assumptions about the type of traffic, e.g., Poisson, service times and the queuing discipline. The process of planning service function chains, creating virtual resources to host network functions and booting them up takes time [28]. Loading the network function software for various VNFs, chaining, acceptance testing, and commissioning need additional time. Initial placements and reconfigurations planned based on calculations at time t, and the state S_t, are actually carried out at a time t_1. In due course, parameters may change and require fresh reconfiguration [29].

Fig. 4 shows the SFC to be placed and the available clouds. Used and installed compute capacities (in integrated units) are shown within the clouds, and so are the used and installed link capacities in M (Megabits) or G (Gigabits) per second. At time t, the assessed end-to-end latency is 20 ms. When the actual placement and activation takes place at time t_1, the latency turns out to be 50 ms. This may cause SLA violation right at the inception and trigger reconfiguration of the chain. When this happens for several service chains, it may lead to a heavy penalty to be paid by the CSP and a loss of customers and revenue for the carrier. When the states of the target clouds are known, the set of least-cost clouds, which give cost and latency below the stated thresholds, can be determined.

Thus, if the state St_1 at the time t_1 can be predicted and the placement is carried out based on this state then the placement remains consistent with the requirements. This is demonstrated by our empirical study given in Section 6.

How is the placement carried out: In an operational CSP set-up as well as the carrier network, a large amount of useful labeled data

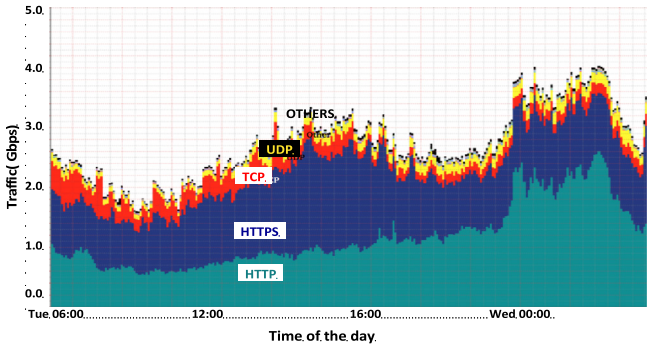


Fig. 5a. Traffic variation on Chicago–Seattle link.

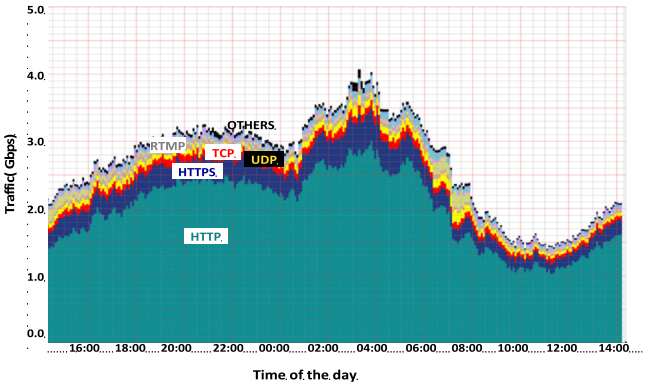


Fig. 5b. Traffic variation on Los Angeles–San Jose Link.

is available, which can be curated for use with supervised machine learning techniques. As the speed, simplicity, and accuracy are of concern, we worked on a prediction technique that could be applied repeatedly for cloud set selection consistent with the objectives of the framework. A review of the literature shows that many supervised machine-learning techniques have been used in cloud computing settings, such as Artificial Neural Networks (ANNs), Bayesian networks, Ensemble classifiers and Support Vector Machines (SVMs). We worked with a number of methods and found interesting results using a well trained and tuned support vector regression (SVR). We discuss the results given by some well-known stock algorithms to show the reason for our choice in Section 6.4. SVR offers the advantage of a unique global minimum as it solves a convex optimization problem. Also, it is amenable to incremental learning. We found that it adapts well to multi-modal cases where the latency is time variant and needs multiple models to fully capture the actual situation. Well-tuned and trained models generalized well from training to the production environment. The results of our experimental evaluation are given in Section 6. For a thorough exposure of SVR, readers are referred to [30].

5.2.2. Time adaptive placement — incorporating temporal variation of traffic in the model

We show through our empirical analysis that taking diurnal traffic variations into account will improve prediction of latencies. *In* carrier networks, there is temporal and spatial variation in traffic demand because of time differences and patterns of use. The amount of traffic flowing through the virtual devices and links varies from place to place and hour to hour. This affects the latency experienced by the subscribers of the carrier's VNS. If the provider over provisions the resources, to meet the surge in traffic in the busy hour, then resources may lie unused most of the time. On the other hand, if enough resources are not provisioned fully in order to reduce the cost of the deployment, then traffic may be lost along with the associated revenue. Figs. 5a and 5b show an hourly variation of the actual traffic on a 100 Gbps link from Chicago to Seattle and 10 Gbps link from Los Angeles to San Jose [31].

The traffic that a carrier routes through the VNFs consists of streams of voice, video, and data with different probability distributions. Each of this traffic varies independently in the time domain. The aggregate traffic in the CSP's network is a composite of all the tenants' traffic and has a complex distribution. The traffic flows continuously as data streams and has properties of big data [32]. In such a dynamically changing and non-stationary environment, the data distribution changes over time, causing the phenomenon of concept drift [33]. The drift is characterized by the change in the density function that is, in turn, reflected by the change in the shape of the traffic distribution or its statistical properties like mean and variance. Thus, the joint distribution p_t of the predictor variables ($\mathbf{X}$) and the labels ($\mathbf{y}$)

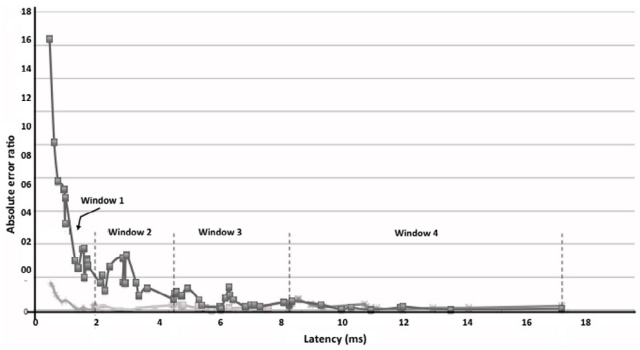


Fig. 6. Comparison of generalization error with an integrated model and FPTV model.

would change dynamically over time such that at time t_0, $t_1, \ldots, t_n$ the following relationship (2) holds for all $\mathbf{X}$.

$$p_{t0}(\mathbf{X}, \mathbf{y}) \neq p_{t1}(\mathbf{X}, \mathbf{y}) \neq \cdots \neq p_{tn}(\mathbf{X}, \mathbf{y}) \tag{2}$$

How do we propose to solve the diurnal traffic variation problem?: The solution that we propose takes care of the concept drift to ensure more accurate traffic predictions. While a single SVR model works well in situations where there is no sizable ambient traffic from other applications and network services. However, SVR by itself does not take care of the time-varying nature of the traffic present on the links from other voice, data, and video applications. To handle this, we incorporate time as a feature by allocating numerical codes to windows.

Researchers have experimented with both fixed and adaptive window methods to handle concept drift in real time situation. In the case of *fixed windows,* the data is segregated into many small windows to have lower overall generalization errors as compared to a single window situation [33]. The utility of fixed window sizes under certain conditions for topological data analysis has been shown by the authors in [34]. A window of a certain minimal fixed size allows learning concepts because the extent of drift is appropriately limited [35]. In Adaptive Windows [36], the window size is changed so that the difference in errors (ϵ), given by a point in two neighboring windows, is bounded by a small value δ such that $\epsilon_t - \epsilon_{t-1} < \delta$.

To achieve a good compromise between prediction accuracy and complexity, we propose a method that has the simplicity of a fixed number of windows and is also flexible to include a variable number of traffic data points depending on the frequency of variations in different windows. Consequently, we call this method fixed-time variable-points (FTVP) window. SVR models are trained, one for each window, to tackle the effect of the concept drift. While even as few as two windows give an improvement in prediction, finding the right number and sizes is a matter of optimization. A larger number of small windows may give more accuracy, but would produce a larger number of models and

would necessitate maintenance of all of them. Using this concept, time is incorporated as one of the features in the training examples. In a sense, each example carries a time-stamp, which makes it a member of a particular FTVP window. When a prediction for a new point is made, the time feature will cause the framework to use the model appropriate for the corresponding time window. In our experiments, this method gives far lower prediction root mean squared error (RMSE) and absolute error ratio (AER) than a single integrated windowless model.

To validate the FTVP concept, we created a trained SVR model using a single window (full integrated dataset) and separately for each of the four selected FTVP windows. In Fig. 6, we show a plot of the absolute error rate versus the latency for both cases. The motivation for using multiple training datasets, using time as one of the predictors, becomes amply clear. The errors, in general, remain more controlled in the FTVP case.

5.2.3. Corrections for short-term traffic variations — incremental learning from new data

In an operational network, the dynamicity of the environment would render the trained predictive models obsolete if the effect of the short-term changes in the traffic is not accounted for. Short-term variations are caused by events like festivals, game tournaments, or rallies. If the effect of short-term changes in traffic is not taken care of, latency prediction and consequent placement decisions may not be correct. Since retraining of all the models would entail prohibitive time and cost, we have used an incremental update of the models. The authors in [33] confirm that the online method can adapt to sudden changes.

Choice of SVR for prediction makes incremental learning easier to understand. In SVR, the support vectors are the only points that determine the decision surface. They also satisfy the Karush-Kuhn–Tucker **(**KKT) conditions [30]. Each new point generated because of the change in traffic is checked for being a support vector. If it is a support vector and improves the overall model for future predictions, then it is included. If this becomes time-consuming, due to continuously generated traffic data, training in small batches speeds up the process. Support vectors can be separately found for each batch of fresh points, and they can be included in the model only if they improve it. Algorithm 2 gives the incremental training algorithm. We see in the next section that this contributes positively to the model empirically.

The initial training process creates a set $S = \{x_s, y_s\}$ of support vectors that decide the decision surface. Algorithm 2 starts with the solution function f(t) at time t in terms of the initial training dataset $T = \{(x_i, y_i), i = 1, \ldots, n\}$ $x_i \in R^n$ and $y_i \in R$. The set of support vectors at this time are $S(t)$. For the time $t+1$ for which the model needs to be incrementally updated each of the new example $\{x_{new}(t), y_{new}(t)\}$ is received in the time window $(t, t+1)$, the algorithm checks if the new point is a support vector. The new support vectors are incorporated in the set $S(t+1)$ if they improve the performance of the model as indicated by reduced mean squared error. Our simulations given in Section 6.6 also support this argument. The simplified algorithm is given below:

Algorithm 2: TRAIN_REAL_TIME (T, x_{new}, y_{new})

```
1: //Initial training set T = ((x1, y1)...(xn, yn))
2: f(t) = A(T)        //Training done at time t
3: f(t) : S(t)        //S(t) is the set of support vectors at time t
4: Initialize S(t+1) to S(t)
5:for all {xnew, ynew} in the window (t, t+1)
6:      if xnew(t) : xs and ynew(t) : ys  // new point is a support vector
7:          S(t+1) = S(t+1) ) ∪ (xnew, ynew)
8:      endif
9: endfor
10: output f(t+1) : S(t+1)      //updated model at t+1
```

The removal of support vectors when the short-term traffic condition that created them has passed will be taken up as future work.

5.3. Cost optimization

5.3.1. Random optimization for cloud selection

An important part of the solution is to select the set of clouds that would be used for placing the VNFs of an SFC such that the total placement cost is the lowest possible, within the budget C_B specified by the carrier, and is consistent with the latency constraints, i.e., $\sum_i l_i \leq L_{th}$ where l_i is the latency within ith cloud, and its link to the next cloud and L_{th} is the threshold given in the SLA. Following Occam's razor, we looked for an algorithm that would be simple and yet effective in meeting the real-time requirements. Algorithms like A-Star are efficient in finding a low-cost walking path from one node to another. Even with one parameter, i.e., the length of the path, its time complexity can degenerate to exponential.

A naïve approach is to search m lowest cost clouds (enough to meet the capacity requirements), one at a time out of total n ($m \leq n$) such that the total cost (in terms of cloud resources and links) is minimized and the latency remains below the given threshold. In large networks, a systematic search like this for the global minimum becomes impractical [37]. The worst case time complexity of this algorithm can be assessed as follows: the search for each next lowest cost cloud requires approximately n lookups, searching m clouds would have the complexity $O(mn)$. Again in the worst case, we would need to look through all the remaining $(n - m)$ clouds to make sure the latency is below the threshold. Thus the complexity is $O((n - m).mn)$ or $O(n^2m - nm^2)$. Selecting just five clouds out of a *hundred* would require 47,500 iterations. In Section 6.8 we compare the randomized cloud search with a modified sequential baseline method to show the usefulness of the adopted technique.

We find that the application of the general theory of optimization by random search gives us good results in the multi-cloud environment. The mathematical treatment of this technique is given in [38]. We have adapted this model to multimodal cases in the presence of constraints [37]. The random search algorithm pursued in this work belongs to the category of Global Optimization. This category of algorithms is useful and efficient for large-scale ill-structured global optimization problems. In contrast with the deterministic methods like branch and bound which guarantee asymptotic convergence to the optimum at the high computational effort, random search algorithms find a relatively good solution quickly and easily. It has been shown that a global optimum can be found with random optimization even if the objective function is multi-modal [39]. Deterministic methods for global optimization are NP-hard, a random search method may be executed in polynomial time [40]. Many of the global random search (GRS) algorithms have the following desirable features because of which they are popular (i) the algorithms are usually easy to construct with guarantee of convergence, even if the objective function is multimodal [40]; (ii) they are insensitive to noise in the objective function; (iii) they are insensitive to the shape of the feasible reason; (iv) they are insensitive to the growth in the dimensionality of the feature set (c). In these cases, it is relatively easier to construct GRS algorithms guaranteeing theoretical convergence. The theoretical basis of general random search is given below. The implementation is shown in Algorithm 3, and the convergence is proven empirically in Section 6.8.

According to [41], the general problem of minimization can be stated in terms of minimization of the objective function f(x) in the feasible region $x \in \mathbf{X}$, if x* is the global minimizer of f(x) or $f(x^*) = \min_{x \in \mathbf{X}} f(x)$. A global minimization algorithm constructs a set of points x_i i = 1…n, in X. A global minimization algorithm is a rule for constructing a sequence of points $x_1, x_2, \ldots$ from the region X, such that the sequence of labels $y_{i=1\ldots n} = \min_{i=1\ldots n} f(x_i)$ approaches the minimum f(x*) as n increases.

To establish the convergence of a global random search, we assume that if x is randomly chosen from within the region X, then f(x*) is a result of some stochastic process. We are presuming a generalized construction of the algorithm where the next point can be chosen from

the entire space. Thus, if $X \subseteq R^d$ and $0 < X < \infty$, $\sum_{j=1\ldots\infty}$ inf $P_j(B(x, \varepsilon)) = \infty$ for all $x \in \mathbf{X}$ and $\varepsilon > 0$, where $B(x, \varepsilon) = \{y \in X : \|y - x\|_2 \leq \varepsilon\}$ and the infimum is over all possible previous points $x_{1\ldots(j-1)}$ and the result of the evaluation of the objective function at these points. P_j are the probability distribution of x_j. Then with probability one, the sequence of points $x_1, x_2, \ldots$ falls infinitely often into any fixed neighborhood of any global minimizer. In other words, if the algorithm is allowed to converge to a global optimum in a finite number of iterations within an acceptance probability, then it will converge with probability one [41,42]. The authors in [38] prove that as long as random sampling does not ignore any region, then the algorithm converges with probability one.

As even for large chains, the number of clouds from which resources are to be taken is not very large; we apply random selection to our problem by selecting at each step a unique set of the desired number of clouds randomly. Accordingly, we repeatedly choose, with replacement, a set M of m clouds from a space N of n clouds (such that m ≤ n) with replacement. If the total cost of the last set is less than the set examined in the last iteration, and the latency is still less than the prescribed threshold, then the algorithm remembers this set. The cost includes that of cloud resources and inter-cloud links. The link costs are usually much larger and ensure locality of clouds while selecting clouds for placement. When the random selection no longer changes the achieved least cost, the process terminates, and the resulting least cost cloud-set is used for placement of the SFC in Algorithm 1. Alternatively, to ensure graceful stop, if the difference between the last two costs falls below a given value, the process can be terminated.

It is appropriate to mention that the total cost and latency of the selected cloud-set places an upper bound on the final figures as eventually more than one VNF may be placed on the same cloud, and all the clouds in the selected set may not be used. As the algorithm iterates over the available clouds, the set M clusters around the minimum. The algorithm converges to the global minimum, with probability one, even in a multimodal case, as long as it does not consistently ignore any of the clouds in the space N. These conditions are met in our implementation. Algorithm 3 gives the details of random selection. The procedure PREDICT_LATENCY has not been separately elaborated as it is based on the SVR model(s) refined for concept drift and short-term changes in traffic as already discussed above.

Algorithm 3: **RANDOM_SELECTION** (***C***, L_{th}, *cv_model, r_clouds*)

```
1: //C: a set of available clouds, cv_model: trained model
2: init small          //contains the sum of costs of the current smallest cost clouds
3: init lat            // lat: latency
4: init iter           //set iterations large enough for convergence
5: while (iter)
6: init r_clouds       // r-cloud array holds final min cost set of clouds
7: //find a set of m unique clouds
8: while (m_clouds not unique)
9:    m_clouds ←a random set of m clouds from set C
10: end while
11: //test set r_clouds still has the lowest cost and lat ≤ threshold
12: call PREDICT_LATENCY  //uses trained and refined models
13: for k = 1, m
14:   lat = lat + lat_k          //initial assessment of total latency
15:   cost = cost + cost_k
16: end for
17: if cost < small and lat ≤ L_th
18:   small = cost
19:   r_clouds ← m_clouds
20: end if
21: end while
```

Algorithm 3 expects CSP data like the available clouds *C* and a trained prediction model *cv_model* and produces a set of 'm' minimum cost clouds to be used for placement by Algorithm 1. The variable *small* represents the smallest total cost of the selected clouds. In line 8–10 a set of *m* unique clouds is selected. Line 12 calls the procedure that predicts latencies for the selected set of clouds. The total cost of the selected clouds is checked against the current minimum cost, and if found to be lower then the vector *r_clouds* is updated with the new set of clouds and *small* with the new lower cost.

5.4. Increasing speed and acceptance ratio of placement

These requirements arise from the dual necessity of real-time usage and agility of the service deployment.

(a) *Speed for real-time usage*

In an operational virtual network service, the cloud service provider needs to monitor latency continuously for avoiding a breach of SLA requirements. Not only the latency and other QoS requirements should be met on initial placement, but also during operation of the service. If the end-to-end latency goes over the stipulated threshold, then the change of placement of VNFs and reconfiguration of the SFC is required. This necessitates the algorithm to be fast in giving optimum SFC placement, migration, and scaling (increasing or reducing the number of instances) decisions so that the network can be dynamically managed. As reported in the literature, ILP based solutions for the placement problem may take a long time (of the order of hours) to converge to the optimum solution [43] making them unsuitable in many situations of dynamic placement.

(b) *Efficiency of placement*

The efficiency of placement refers to successful placement rate (also called the acceptance rate) and reconfiguration of chains consistent with SLA requirements. It is important for this rate to be high since frequent failure to place and reconfigure chains according to the requirement may lead to the carrier not being able to handle customer requests.

5.5. Combining the elements of the framework

The placement strategy described above has been implemented in a placement framework called the P-ART framework. The main modules of P-ART are as shown in Fig. 7 along with the relationship with the algorithms discussed.

The framework allows CSP and carrier policies to be stored as well as the means for them to communicate with the framework. The instant state of a cloud consists of the used capacities of virtual compute, storage and networking resources. For each placement request, the management and monitoring module produces a success or a failure report. A brief description of the modules is as follows:

SVR Training and Windowing: This part takes the integrated dataset and breaks it into a separate dataset for the specified number of windows. It then trains one model for each window applying the FTVP methodology discussed above. Short-term changes are incorporated through incremental training. These predictions are used by the prediction module to give an assessment of latencies at the time of placement.

CSP Policies: Through this module, the cloud service provider (or a multi-cloud broker) enters the cloud configuration data, installed and used cloud capacities, installed and used link capacities as well as tariffs for resources.

Carrier Policies: This module accepts client's requests for changes in service chain placements, types of virtual functions and inter-function traffic rates. Operative parts of the tenants' SLAs, including latency, threshold, and cost budgets are also stored. Carrier privileges are also recorded in the database.

Prediction module: The prediction module uses the correct model for prediction of latencies at the time of activation of the chain. It predicts

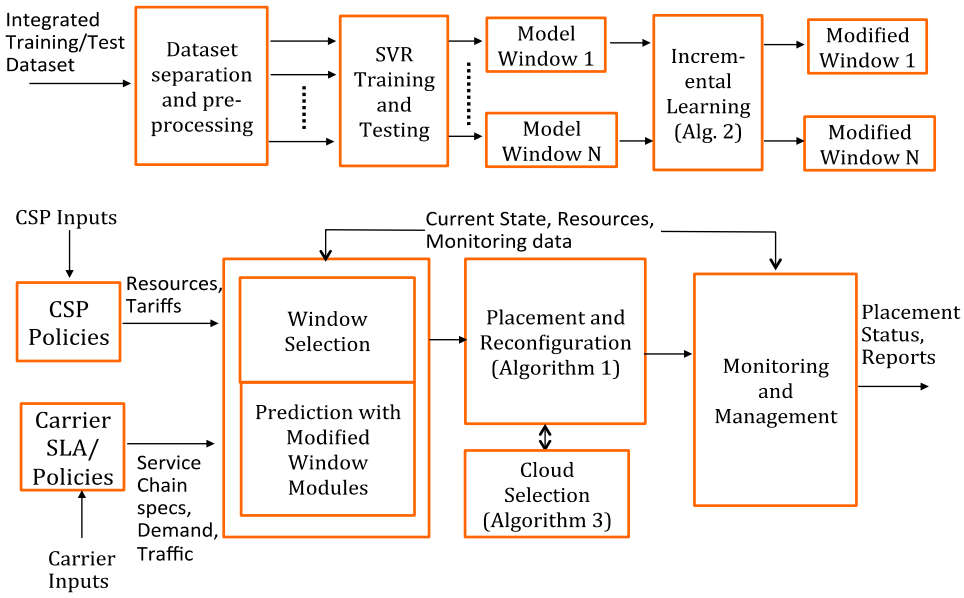


Fig. 7. The P-ART placement framework.

the latencies among clouds at the time an SFC would be actually placed and activated.

Placement and Reconfiguration Module: This module carries out placement, scaling, and adaptation to the changed State of the environment. Heuristics for placement has been devised to work fast and converge to a set of clouds close to the minimum cost and latency below the threshold. If a placement is successful, it gives the end-to-end latency and cost.

Monitoring and Management Module: This module keeps an inventory of the resources used, the status of performance parameters and the state of the cloud environment. If placement is successful, it gives the end-to-end latency and the cost. Online monitoring reports are part of the future extension.

6. Evaluation of the framework

We evaluated the P-ART framework to confirm the validity of all the sub-systems incorporated, viz., model training and generalization, prediction and its refinement, cloud selection for placement, speed and acceptance ratio of placement. To keep evaluation close to reality and to cross-verify results, datasets used for training and testing were generated in two ways: *simulation* using a queuing-theoretic model and an actual *implementation* on CloudLab [9].

6.1. The experimental set-up for evaluation

In our experiments, we use multiple instances of the VNS using one SFC with 5 VNFs introduced in Section 5 (Fig. 3). As we shall see in Section 6.8, the method scales well for bigger chains with thousands of virtual functions. The traffic entering the aggregation switch (VNF1) is divided into two streams, one going to one of the Provider Edge (PE)-routers (VNF2 or VNF3) depending on the carrier's traffic routing policies. For instance, the policy may route traffic from different geographical areas through different paths. All the traffic passes through one of the instances of BNG (VNF4) where in practice, the flow accounting will take place for billing purposes. The traffic is then routed to P-Router on route to the destination. The end-to-end latency of the chain would be the greater of the latencies of the two paths VNF1-VNF2-VNF4-VNF5 and VNF1-VNF3-VNF4-VNF5.

In the experiments reported here, the CSP domain consists of 10 clouds. However, we also tested the random selection algorithm for a larger number of clouds, and the results have been discussed in Section 6.6. Without the loss of generality, we generate the link capacities randomly from the chosen set of realistic capacities. In our experiments, we choose from the set L= [0.016, 0.064, 0.100, 0.155, 0.622, 2.5] (in Gbps). All links are presumed to be bi-directional. The compute capacities of the VMs hosting VNFs have been taken as a single consolidated figure for processor, memory, and storage. An example of such a usage is Amazon EC2 where, for instance, t_2, the medium virtual machine provides two virtual CPUs, 4 GB storage and elastic storage. In our experiments, the categories defined are as shown in Table 2.

Table 2
Categorization of server resources.

Integrated capacity	vCPUs	Memory	Storage
1	1	1 GB	Flexible
2	2	2 GB	Flexible
4	4	4 GB	Flexible
6	4	8 GB	Flexible
8	8	8 GB	Flexible
10	8	16 GB	Flexible

6.2. Selection of features for training the prediction models

Considering the importance of the selection of predictor variables, due attention was given to this aspect. Too many features can make prediction models complex, increase the training time and make test errors worse. Further, selecting a good set of features, out of all the features generated, improves the accuracy of prediction and speed of processing. Cross-validation error has been used to guide feature selection for our prediction models in SVR. Features that do not give an improvement in terms of lower overall errors (indicating better prediction) were removed from the initial feature set. We settled on the set of features given in Table 3. Further analysis, to include other variables that are not highly correlated with the existing ones, but may reduce the cross-validation error, is left as future work.

As seen in Table 3, the feature space is represented by $\boldsymbol{X} = [x_1, x_2, x_3, x_4, x_6, x_7, x_8]^T$ and corresponding labels $\boldsymbol{y}$. The equipped physical compute, and storage capacities of a server govern the number of VMs that can be created on it and correspondingly the number VNFs that can be hosted. VMs on the same PM may cause interference in each other's operation because of shared resources which may lead

Table 3
Predictor variables and output label.

Predictor variables		Label (output)
x_1	Origin cloud compute installed capacity	y: Latency (ms)
x_2	Destination cloud compute installed capacity	
x_3	Link installed capacity (Gbps)	
x_4	Link used capacity (Gbps)	
x_5	Origin cloud compute capacity used	
x_6	Destination cloud compute used capacity	
x_7	Window #	
x_8	The distance between the origin and destination clouds	

to delays. As far as the links are concerned, each additional Gbps of equipped capacity does not give the same increase in traffic carrying capacity. The amount of traffic that can actually be carried depends on the grade of service required. Total ingress traffic depends on the number of served subscriber clusters. The end-to-end latency depends on the traffic, requiring this feature to be included. We have seen in Section 5 that traffic is dependent on the time of the day. We discussed the number of windows and its relationship with the complexity of the model. The increasing window number is indicative of the increasing time of the day. While the number of windows is a parameter in the evaluation, we obtained good compensation of concept drift with four windows as indicated by the results.

6.3. Obtaining training datasets

We were cognizant of the fact that if a model has been trained with the adequate, realistic dataset, it will generalize well in the production environment. For a more thorough evaluation of the model, we use two methods for generating datasets. One dataset was obtained through simulation of inter-VNF traffic flows and the other through actual implementation of the service chain on CloudLab. The details are as follows.

6.3.1. Inter-VNF traffic flow simulation

Carrier networks carry all kinds of traffic: voice, data, and video. Some of these applications are real-time, and their packets have higher priorities. When queues build up at link or router buffers, the higher priority traffic may pre-empt lower priority traffic. It follows that different types of traffic will experience different delays. The delay model shown in Fig. 8 takes care of all the important delays. Queuing delay in the links is the variable part of the end-to-end delay and depends on the network load. Propagation delay is the time required by the signal to travel on the link from one VNF to another. This delay depends on the media and is proportional to the length of the link, approximated by the distance between clouds. The other prominent delays are processing delay in the clouds, queuing delay in the virtual machines, and transmission queuing delays on the link. Intercloud simulation was carried out covering all significant delays.

The total time spent by voice and data packets in the network can follow any distribution. Following the conclusion in [44,45], we have assumed an $M/G/1$ queuing system of infinite capacity with non-preemptive priority. The traffic load is varied to imitate the pattern of the actual traffic. A C++ routine generates the dataset that incorporates all the parameters described above. The dataset was normalized to keep the numbers comparable. This will prevent any feature from overpowering others in the model and avoid biases.

6.3.2. Cloudlab implementation

CloudLab is a "meta-cloud" that has been implemented by the University of Utah, Clemson University, the University of Wisconsin, Madison, the University of Massachusetts Amherst, Raytheon BBN Technologies, and the US Ignite for researchers to build their own clouds for experimentation [45]. The software stack that manages CloudLab is based on Emulab. The infrastructure at Utah, Wisconsin

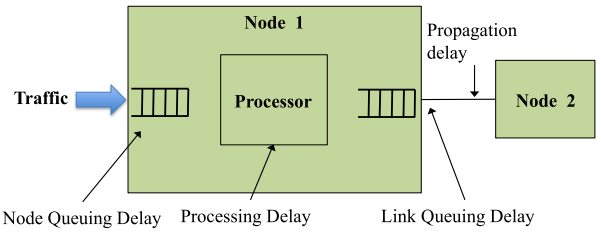


Fig. 8. Traffic delay model for data generation.

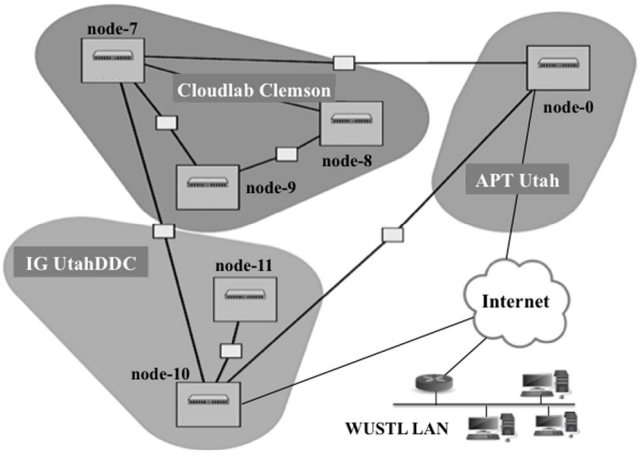


Fig. 9. The CloudLab implementation.

and South Carolina is interconnected with nationwide and international infrastructure from Internet2, so it has been possible to extend, software-defined networks right to every host. The CloudLab set up created for this study is shown in Fig. 9.

The data collection process involves traffic being routed from a host on the WUSTL (Washington University in St. Louis) LAN through the Internet to the CloudLab nodes. Thus the test traffic goes with the live traffic on the Internet and provides real-life traffic conditions. Nodes 0, 7 and 10 are the transit points for traffic at APT Utah, Clemson University and IG Utah DDC (InstaGENI Rack in Downtown Data Center) clouds, respectively. The distance from the host at Washington University in St Louis to each of these were IG Utah DDC (800 miles), Clemson University (1950 miles) and APT Utah (800 miles). The VNFs are presumed to be hosted as follows: VNF1 on node11, VNF2, and VNF3 on Node 10, VNF4 on Node 7 and VNF5 on Node 9. Delays on the link from WUSTL to the CloudLab depended on the traffic on the Internet. Within CloudLab the delays were varied by loading the links with different amounts of traffic. Various delays were recorded as part of the training data. A snapshot of part of one of the training sets is shown in Table 4.

6.4. Selection of the machine learning model

There are quite a few AI techniques, involving machine learning, that are potentially applicable to the problem of detection and localization of fault and performance anomalies. Models with a single layer of non-linearity, e.g., a neural network with one hidden layer, are referred to as shallow structures or shallow machine learning architectures and those with more than one layer of non-linearity as deep structures or deep learning architectures. Shallow models with linear hypothesis may have O(n) prediction time complexity and training time of O(l^2+n^3) where l denotes the size and n the degree of the dataset, but approximation errors are large for the high dimensional and large volume of data that are usually associated with FP problem. With non-linear hypothesis

Table 4
An extract of the integrated training dataset.

o_cap (x_1)	d_cap (x_2)	link_cap (x_3)	link_used (x_4)	o_used (x_5)	d_used (x_6)	window (x_7)	link_len (x_8)	latency y
1	1	0.622	0.2488	0.4	0.4	4	0.4	2.2482
2	6	2.5	1.25	1	3	4	0.4	1.93968
1	1	0.3	0.12	0.4	0.4	1	0.1	0.549477
2	1	0.064	0.0064	0.2	0.1	2	0.4	7.87455
4	1	0.016	0.008	2	0.5	2	0.2	11.0425
1	2	0.016	0.0032	0.2	0.4	2	0.4	10.9699
1	1	0.622	0.1244	0.2	0.2	3	0.4	4.2971
1	1	0.3	0.21	0.7	0.7	4	0.1	1.9999
1	1	0.3	0.09	0.3	0.3	1	0.1	1.59822
4	8	0.3	0.18	2.4	4.8	1	0.1	0.95361
6	1	0.064	0.0064	0.6	0.1	2	0.4	7.45232
1	1	0.3	0.03	0.1	0.1	1	0.1	1.02085
4	6	0.1	0.04	1.6	2.4	3	0.4	4.78636
2	1	2.5	0.5	0.4	0.2	4	0.2	3.15644

Legeng: origin cloud installed capacity (o_cap), destination cloud installed capacity (d_cap), link installed capacity (link_cap), origin cloud used capacity (o_used), destination cloud used capacity (d_used), length of the link between origin and destination (link_len).

space and kernel trick, the approximation errors may be smaller at the cost of higher complexity of the training time which is $O(l^3 + l^2n)$ and prediction speed of $O(ln)$. Of the prevalent shallow machine learning architectures, Support Vector Machines (SVM) and Random Forest (RF) are considered useful for diagnostic applications [46]. Another supervisory technique, Bayesian Network (BN), has been applied to fault management in the industrial settings. We will discuss below the analysis that was carried out to finalize the model [47].

Size of Training Dataset: The size of the available training dataset governs the choice of the machine-learning algorithm. How much data is enough depends on the number of features and the non-linearity in the relationship of features and labels among others. If the dataset is small, one may choose high bias and low variance classifiers like Naïve Bayes as compared to the low bias and high variance classifies like kNN to avoid overfitting. When the training dataset size is large, low bias and high variance classifiers give a lower asymptotic error.

Number of Parameters: Most machine learning algorithms are associated with some parameters and hyperparameters. Parameters of an algorithm are internal to it and their values affect how the algorithm behaves. They are usually learned at the time of training of the model. The value chosen for these parameters may affect the accuracy with which the model predicts. Support vectors of the SVM algorithm are an example of a model parameter. Hyperparameters are normally external to the algorithm. They need careful tuning to get good accuracy from the model. An example is the C hyperparameter in SVM. Even though having many parameters or hyperparameters typically provides greater flexibility, training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings.

Number of Features: If the number of features is large then the dataset is said to be high dimensional. With high dimensional dataset, we need more data to train the model. Increase in size of the dataset affects different algorithms differently. The complexity of some machine learning algorithms may rise exponentially in such cases. The training time may become too long for the model to be used in real-time applications.

Learning Process: The learning process of a model may be supervised or unsupervised based on whether labels are available or not. Since the labels indicate the ground truth, we know how our trained model should behave. In unsupervised learning, the data is unlabeled, so the model learns the inherent structure in the data. If there is some labeled data and a lot of unlabeled data, then we may use semi-supervised learning in which the labeled data can be used to improve the accuracy of the model built using unlabeled data . Another thing to note is that we are predicting latency values which vary in a continuous range. This would, therefore, call for a regression model as against a classification model.

Table 5
Comparative study of machine learning algorithms.

	Corr. coeff.	Mean absolute error	RMS error	Relative absolute error (%)	Root relative squared error (%)
Random Forest	0.8639	1.1881	2.4219	33.6077	50.3668
SVR	0.8610	1.2426	2.5048	35.4465	52.8385
KNN	0.8007	1.469	2.9681	41.9043	61.7248
MLP	0.8015	1.9317	2.9405	55.103	61.1514
Gaussian	0.5714	2.7523	3.9340	78.5130	81.8128

We need to understand the requirements of the problem to pick the right algorithm for the application. In our case, it is important that the model works in real-time or near real-time. This is possible if the placements and reconfigurations are fast. The model should be fast to train and update with real-time information. This requires models to be generally simple, with controlled dimensionality and a manageable number of hyperparameters to tune. Additionally, some models may not be suitable for online training.

Keeping the above in view, we compared a few suitable stock methods to decide on the one that we would include in our model. The models were created and tested on Weka [48]. In each case, the models were tuned for good parameter values, and a 13-fold cross validation was used. We discuss the methods briefly followed by a comparison of their performance in Table 5.

Random Forest is a supervised method which is robust yet simple to use. It provides good results in many situations. It does not have many hyperparameters to tune, the useful ones being the number of trees and the maximum number of features to be tried in each tree. Despite their flexibility, random forest does not support online learning. Retraining by rebuilding the trees when new examples are introduced takes time. The maximum depth of each tree has been set at unlimited. The number of iterations or number of trees is set as 100.

Support Vector Machine (SVM) is a supervised learning algorithm. The regression version of SVM, which is designated SVR or Support Vector Machine for Regression (SMOReg), gives good accuracy and can work with high dimensional data, which is not linearly separable. Parameter values that obtained for good results are C = 200, $\gamma = 0.01$, $\in$ = 10E-8, RBF Kernel.

K-Means is an unsupervised model and has been included for comparison here. In this, k data points are chosen, and data is divided into clusters with each example going with the nearest data-point. Then, centers of the clusters are converted, and the process repeats until convergence. The result depends on the initial choice of the points, and the global minimum is not guaranteed.

Multi-layer Perceptron (MLP) are neural networks with at least three layers of neurons — an input, a hidden and an output layer. These layers are connected in the form of a directed graph between the input and the output layers. It is also called a feed forward network. An MLP uses backpropagation as a supervised learning technique. Some of the parameters include N (the number of epochs for training) taken as 500, E (the number of consecutive increases of errors allowed for validation before terminating the training) fixed at the default of 20 and L(the learning rate) taken as 0.3.

Gaussian processes are a supervised learning technique and generalization of Gaussian probability distribution. Gaussian distributions are governed by stochastic processes and describe random variables. A Gaussian distribution is fully specified by its mean and covariance matrix. In a similar manner, a Gaussian process is specified by a mean and a covariance function. Some of the parameters are L (the level of Gaussian noise) taken at the default value of 1 and K (the Kernel to use) taken as PolyKernel.

Table 6
Training error.

=== Evaluation on training set === === Summary ===	
Correlation coefficient	0.861
Mean absolute error	1.2426
Root mean square error	2.5408
Relative absolute error	35.4465%
Root relative squared error	52.8385%

Table 7
Test error.

=== Evaluation on training set === === Summary ===	
Correlation coefficient	0.7304
Mean absolute error	1.8895
Root mean squared error	2.5469
Relative absolute error	63.5334%
Root relative squared error	71.5849%
Total Number of Instances	56

Using the root mean square error as a good indication of the appropriateness of the algorithm for the datasets used we see that Random Forest gives the lower error followed by SVR. Taking into account our requirement of online updates, we chose to implement SVR.

6.5. Prediction model tuning and testing

In the SVR models, three hyper-parameters, viz., ∈, C, γ need attention. Tuning these hyper-parameters is one of the main challenges in improving the predictive accuracy of an SVR model. The γ parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. With a small γ, the model cannot capture the complexity or "shape" of the data. If γ is too large, the radius of the area of influence of the support vectors only includes the support vector itself, and no amount of regularization with C will be able to prevent overfitting. The constant C determines the tradeoff between the flatness of f and the amount of error allowed above ϵ. A low C makes the decision surface smooth; a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. Most researchers have followed a standard procedure in using a grid search [9] to determine the appropriate values. Some of the results are given in Table 5. A number of runs narrowed down the parameters to $C = 1 \times 10^{-2}$ and $\gamma = 1$. The cross-validation error for this combination was the lowest at 7.84295×10^3. It is worth mentioning that with system decided settings when the built-in tuning feature is allowed to choose the parameters; the loss is higher at 2.21345×10^4. The grid search has, in this case, resulted in better hyper-parameter values.

The basic idea of using latency prediction is to improve the placement of virtual functions at a future time. This will only work if the predictive model produces good predictions of latency. With the Weka tool, SVR with RBF Kernel with the hyper-parameters set at C=10, $\in = 0.4$ and 20% hold-out for cross-validation, we get the errors shown in Tables 6 and 7. It can be seen that both the training and test RMSEs are low indicating good performance. In the classical case, test errors would be slightly higher than the training errors. A lower test error may indicate overfitting or biases in the dataset. These can be overcome by curating the training dataset.

A comparative plot of training and test error ratios (defined as *prediction_error/acutal_latency)* is given in Fig. 10. It can be seen that the model training errors are low and generalize well with the test data.

6.6. Refinement of latency prediction by compensating concept drift

The FTVP method for handling the concept drift in telecommunication traffic was presented in Section 5.2. This method brings in the sense of time in the datasets. Most researchers working with predictive model do not include time as a feature. In our experience, including time as a feature affects the predictions positively. We divided the data into windows of equal time blocks, which give variable data ranges. The window# is the feature (x_7) in the training dataset and has a direct relation with the time as increasing number relates to increasing time. All the time-related observations were divided into four windows. A sample from each of these is given in Fig. 11.

The data in different windows have different characteristics as shown by the mean and standard deviation in Table 8:

SVR with separate window models gives much better predictions on new data-points falling in those windows. Comparison of latency prediction and error ratios for each window and full dataset is given in Fig. 12(a) through (h).

Table 9 summarizes the mean absolute errors and RMSE for the full (integrated) dataset and the window-based model. In the integrated

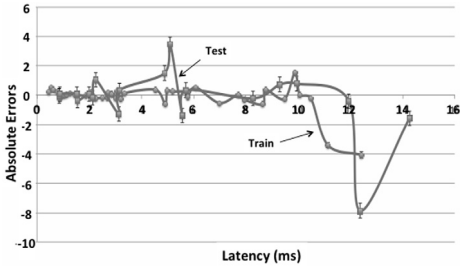


Fig. 10. Training and test error ratios (with standard error bars).

Window.1

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0.3	0.24	0.8	0.8	1	0.1	0.7838
1	4	0.3	0.24	0.8	3.2	1	0.1	0.7838
6	4	0.3	0.06	1.2	0.8	1	0.1	0.8529
6	4	0.3	0.09	1.8	1.2	1	0.1	0.8631
2	1	2.5	0.75	0.6	0.3	1	0.2	1.0717
4	8	0.3	0.24	3.2	6.4	1	0.1	1.1176
2	1	2.5	1.25	1	0.5	1	0.2	1.1345
1	2	0.3	0.15	0.5	1	1	0.1	1.1458
1	4	0.3	0.15	0.5	2	1	0.1	1.1458
2	1	2.5	1.5	1.2	0.6	1	0.2	1.1893

Window.2

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
2	1	0.064	0.0064	0.2	0.1	2	0.4	7.8746
6	1	0.622	0.311	3	0.5	2	0.6	7.8994
2	6	0.155	0.124	1.6	4.8	2	0.6	7.9167
4	1	0.016	0.0096	2.4	0.6	2	0.6	7.9177
1	1	0.622	0.4354	0.7	0.7	2	0.4	7.9403
2	1	0.064	0.0128	0.4	0.2	2	0.4	8.0748
6	1	0.064	0.0256	2.4	0.4	2	0.4	8.1421
6	1	2.5	0.5	1.2	0.2	2	1	8.2461
1	6	0.016	0.0112	0.7	4.2	2	0.2	8.2672
6	1	2.5	1.25	3	0.5	2	1	8.3107

Window.3

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0.622	0.3732	0.6	0.6	3	0.4	4.895
6	4	0.1	0.05	3	2	3	0.2	4.9074
6	4	0.1	0.08	4.8	3.2	3	0.2	4.9327
4	6	0.1	0.05	2	3	3	0.4	4.982
1	6	0.016	0.0048	0.3	1.8	3	0.2	4.9872
4	1	0.016	0.0064	1.6	0.4	3	0.2	4.9972
1	6	0.064	0.032	0.5	3	3	0.2	5.0935
1	6	0.064	0.0512	0.8	4.8	3	0.4	5.0935
6	2	2.5	0.5	1.2	0.4	3	0.4	5.1221
6	1	0.622	0.1244	1.2	0.2	3	0.6	5.1576

Window.4

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
2	6	0.155	0.1085	1.4	4.2	4	0.4	2.7246
6	1	0.622	0.4354	4.2	0.7	4	0.6	2.7258
6	1	2.5	1	2.4	0.4	4	0.2	2.7603
4	4	0.155	0.093	2.4	2.4	4	0.6	3.1406
2	1	2.5	0.5	0.4	0.2	4	0.2	3.1564
1	4	0.155	0.031	0.2	0.8	4	0.2	3.2175
6	2	0.1	0.08	4.8	1.6	4	0.4	3.2716
1	1	0.622	0.3732	0.6	0.6	4	0.2	3.2845
2	1	2.5	1	0.8	0.4	4	0.2	3.2936
2	6	0.155	0.093	1.2	3.6	4	0.6	3.3156

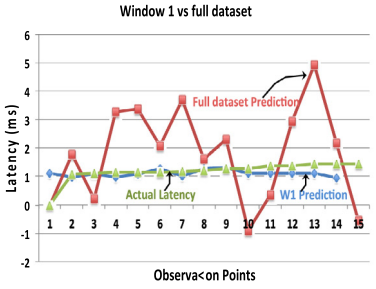
Fig. 11. Extract of FTVP windows.

Table 8
Probability distribution parameters in different windows.

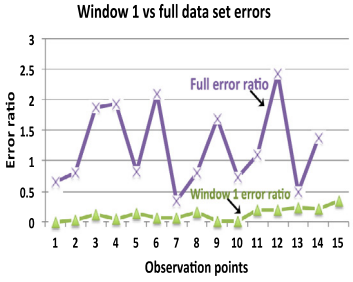
Window	1	2	3	4
Latency range	(1.824–0.422)	(27.683–7.452)	(7.317–4.131)	(4.216–1.869)
Mean	1.083	11.834	5.366	2.773
Standard deviation	0.425	4.848	0.797	0.588

Table 9
Errors with integrated and multiple models.

	Full dataset	Window 1	Window 2	Window 3	Window 4
Mean absolute error	3.2279	0.3698	0.4613	0.7342	2.5248
Root mean squared error	4.5869	0.4283	0.5515	0.9102	2.9353

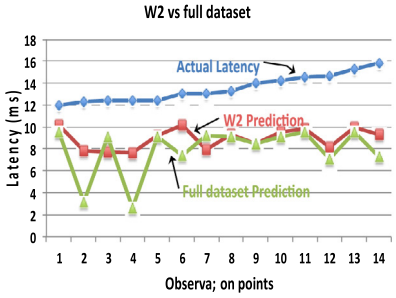


(a) Comparative Window 1 and full dataset performance

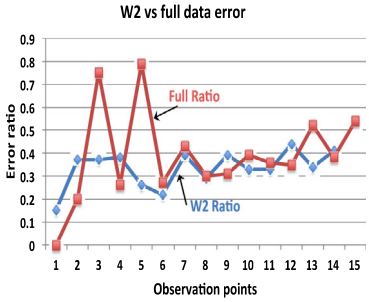


(b) Error ratios for Window 1 and the full dataset

Window 1 RMSE =0.06, full model RMSE =0.47

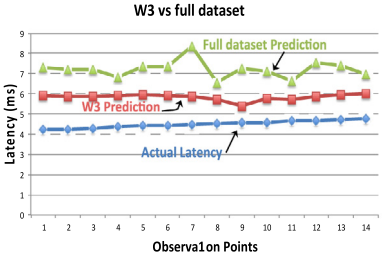


(c) Comparative Window 2 and full dataset performance

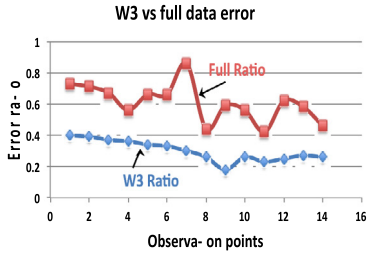


(d) Error ratios for Window 2 and the full dataset

Window 2 prediction RMSE=1.27, full model prediction RMSE = 1.62

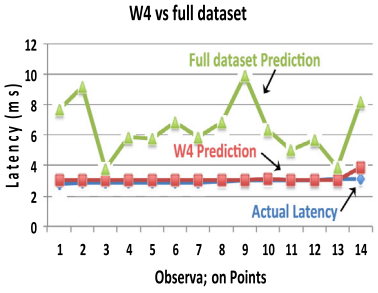


(e) Comparative Window 3 and full dataset performance

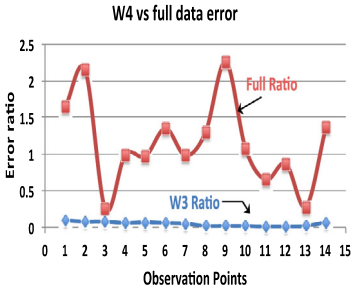


(f) Error ratios for Window 3 and the full dataset

Window 3 prediction RMSE=0.36 full model prediction RMSE = 0.74



(g) Latency prediction by a model trained for Window 4 and that by full dataset



(h) Error ratios for prediction by the model trained for Window 4 and with the full dataset

Window 4 model prediction RMSE = 0.042 Full model RMSE =1.01

Fig. 12. Comparison of performance window-based integrated models.

Table 10
Performance of SVR before adding new support vectors.

Support vectors before online update				Performance before online update	
SV#	Actual latency	Predicted latency	Error	=== Evaluation on test set ===	
50	5.713	5.379	−0.334	Correlation coefficient	0.8742
51	7.452	5.233	−2.219	Mean absolute error	1.2677
52	3.111	3.152	0.041	Root mean squared error	1.7366
53	1.531	2.785	1.254	Relative absolute error	47.3488
54	5.572	4.625	−0.947	Root relative squared error	49.2994
55	5.771	5.298	−0.473	Total Number of Instances	55

Table 11
Performance of SVR after adding new support vectors.

Support vectors after online updation				Performance after online updation	
SV#	Actual latency	Predicted latency	Error	=== Evaluation on test set ===	
50	5.713	5.379	−0.334	Correlation coefficient	0.8816
51	7.452	5.233	−2.219	Mean absolute error	1.2014
52	3.111	3.152	0.041	Root mean squared error	1.6797
53	1.531	2.785	1.254	Relative absolute error	44.5651
54	5.572	4.625	−0.947	Root relative squared error	47.9109
55	5.771	5.298	−0.473	Total Number of Instances	60
56	3.111	3.374	0.264		
57	0.605	2.424	1.820		
58	3.345	3.190	−0.155		
59	3.315	3.579	0.064		
60	10.259	10.199	−0.060		

model validation was done with 20% of the data points separated as a test set. For each window model also cross-validation was done with separate test sets. It can be seen that errors are less in a separate model for each widow compared to predictions made using integrated dataset.

6.7. Incremental update of models to compensate for short-term variations in traffic

We tested an incremental update of the trained models, with support vectors generated during VNS operation, while the trained model was in use. The result of initial training is given in Table 10, and after the introduction of separately generated support vectors, the results improved as shown in Table 11. We can see that both the mean absolute error and the RMSE decrease when new support vector points are learned online. Before the addition of new support vectors, the RMSE was 1.74; while after addition, it reduced to 1.68, which along with other measures of errors show an improved model.

6.8. Cloud optimization with iterative random selection

The principle and methodology of random selection of clouds for placement of VNFs have been discussed in Section 5.3 In one trial, a total of 50 experiments were conducted with 1500 and 1700 iterations each. The minimum possible cost was 51 units, and latency threshold was set at 150 ms. In the former case, 98% of times the minimum cost of 51 units was reached (Fig. 13a) with a latency of 137 ms. In the 1700 iteration case, the minimum cost clouds were selected with the latency below the threshold in all cases (Fig. 13b).

In another trial of 5000 experiments, 50 each with the number of clouds increasing from 10 to 100 in steps of 10 and iterations from 500 to 2000, the convergence rate is as shown in Fig. 14. Somewhere between 1500 and 2000 iterations, the algorithm converges to the minimum cost in 100% cases. This is an order of magnitude improvement over the exhaustive search described above.

We implemented as the baseline a variation of the sequential method, which we call modified-sequential (M-sequential). In this method, the first set of lowest cost clouds were sequentially selected from a set of 100 clouds without replacement. This ensures the lowest cost. However, if the total latency of the selected cloud was more

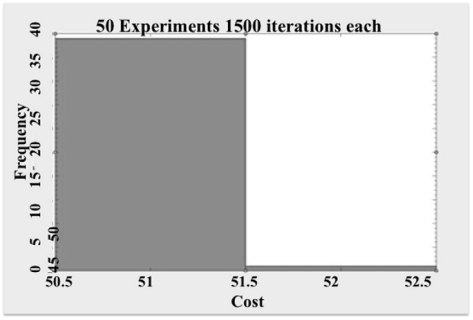


Fig. 13a. 50 experiments with 1500 iterations each.

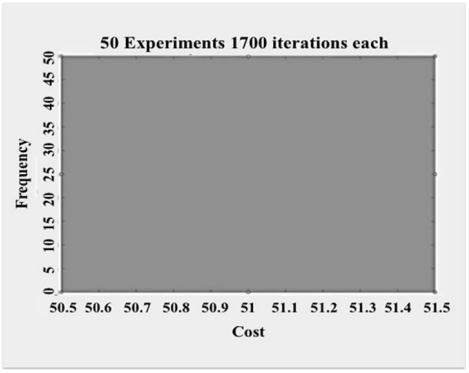


Fig. 13b. 50 experiments with 1700 iterations each.

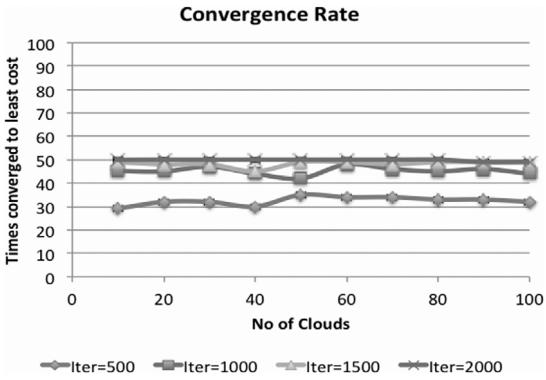


Fig. 14. Number of convergences in 50 experiments.

than the given latency threshold, then the highest latency cloud was removed from the selected set, and a search was made for the next lowest cost cloud. The search stopped when a set of lowest cost with latency below the given threshold was found.

Fig. 15 shows the number of iterations required to achieve the target latencies (from 100 to 160 ms) for both the randomized and M-sequential algorithms. We see that the M-Sequential takes from 34% to about 67% more iterations than randomized. Fig. 16 gives the final latencies achieved in the number of iterations for which the algorithm was run (as shown in Fig. 15). From these, we can conclude that the randomized algorithm performs better than the baseline both in terms of the number of iterations and latencies achieved in selecting the required set of clouds for placement.

6.9. Speed and efficiency

It is important for dynamic rescaling that the designed placement strategy is able to carry out a large number of placements within an

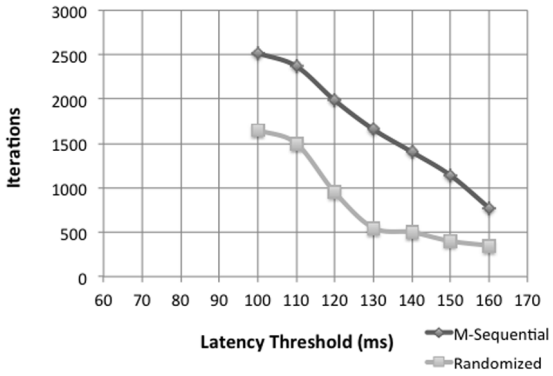


Fig. 15. Number of iterations required by randomized and M-Sequential to achieve latency below the threshold.

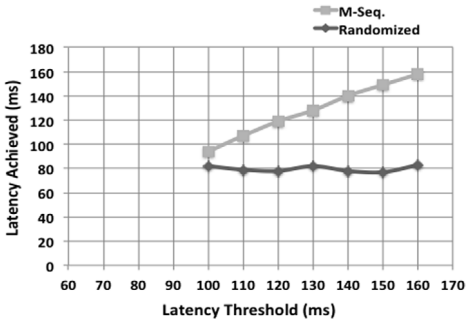


Fig. 16. Latencies achieved by randomized and M-Sequential in the number of iterations shown in Fig. 15.

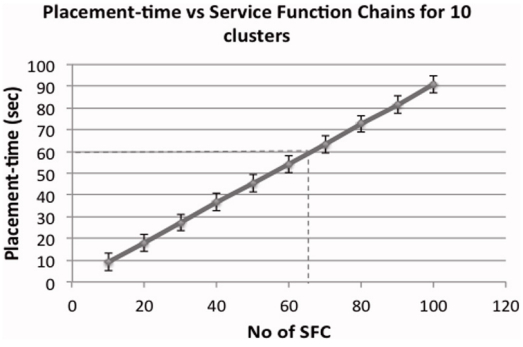


Fig. 17. Placement time vs. No of SFCs.

acceptable time period. A slow placement algorithm would not be able to respond fast to the changing network situation or a tenant's new request. Changes made too late may not be suitable, and may actually be detrimental to the health of the network, as by that time the situation would have changed. On the other hand, if at a future time, maintaining the required performance does not need all the resources that have been deployed, then not descaling would use up a higher amount of resources leading to higher expenses. For the training time of SVR, various assessments of complexity in the range $O(n^2)$ to $O(n^3)$ are available in the literature. According to [29] the complexity is $O(\max(n, d) \min(n, d)^2)$ where d is the size of the feature set. If n is much larger than d, then it can be approximated to $O(nd^2)$. However, the time complexity of the search is linear. It took about 1.19 s to train with 2720 examples in Weka and 0.76 s in MATLAB. For speed of placement, we tested with 10 clusters, each requesting 10 to 100 SFCs of 5 VNFs each. Thus, the number of VNFs was varied from 500 to 5000. We observe that the algorithm is able to place up to 3000 VNFs in about 1 min (Fig. 17).

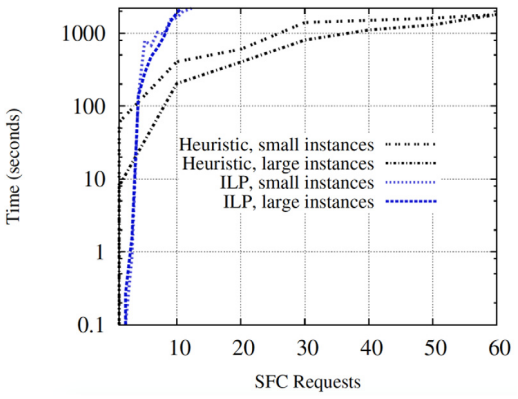


Fig. 18. Placement time reported in [49].

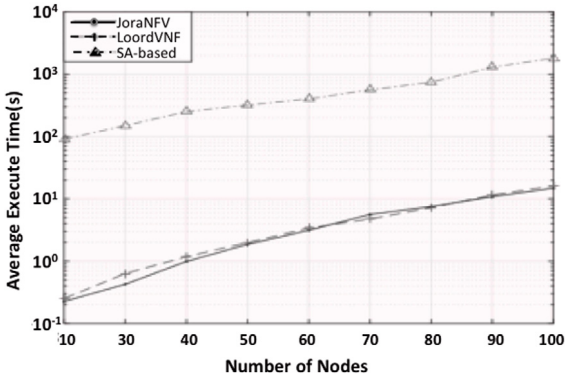


Fig. 19. Average placement time reported in [50].

To see how the speed of the proposed method compares with the placement speeds obtained in other works we see the work done in [49]. The two methods have been performed under different conditions and are thus not strictly comparable. However, we do get the general idea of the behavior of the methods. From Fig. 18, we see that in case of up to 20 SFCs the ILP solution is able to find a solution but the author reported average time is 8 min and 41 s and that of heuristic 1 min and 21 s. For the case of 60 SFCs, the ILP model takes unduly longer times (>48 h for ≥18 SFCs). The heuristic was able to give a solution in less than 30 min. For small instances, 40 SFC requests (with 75 network functions per request or a total of 3000 functions) take about 1000 s.

A comparison has also been made with results obtained by a completely different technique presented in [50]. The authors have carried out joint optimization of resource allocation in NFV (JoraNFV). The authors assume that the number of VNFs can be 3, 4 or 5. Taking the example of a 5 VNF SFC and medium traffic, the authors conclude that their method works faster than CoordVNF [51] and a simulated annealing approach [52].

The coordinated NFV-RA is formulated as mixed-integer linear programming (MILP). And we propose a heuristic based two-stage approach to get the near optimal solution. For ten units of traffic, the number of instances deployed are about 7 for JoraNFV, 10.5 for CoordVNF and 7 for the SA method. For a 90 node network, the JoraNFV and CoorNFV take 10 s to place an SFC while SA takes about 2000 s Even if we assume a linear increase in time taken, for 3000 functions/instances JoraNFV will take 4285 s (Fig. 19).

ILP based solutions for a large number of VNFs are slow, even with efficient solvers. Researchers in [27,53] have carried out VNF placement of different configurations using ILP method. In [27], the authors have reported that ILP takes 2.3, 4.0 and 7.2 h for 10, 30 and 50 functions. In [53], the authors have tried to solve ILP for large networks

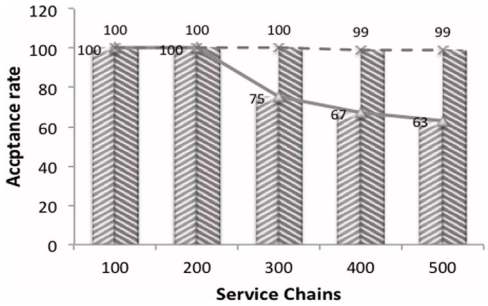


Fig. 20. Acceptance rate vs. number of SFCs.

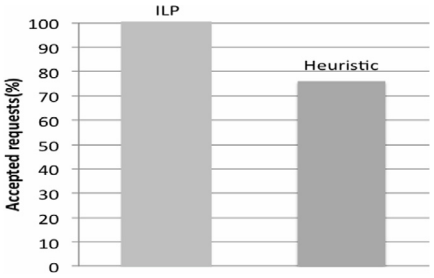


Fig. 21. Acceptance % reported in [54].

(60 SFC with 4 VNFs and 30 instances, each, i.e., 7200 VNF instances) but for more than 18 SFCs the time taken is more than 48 h. The authors have suggested heuristics to find an acceptable solution within reasonable time limits. Thus, [28] suggests using Genetic Algorithm with which 200–700 functions are placed in 8–13 s. In [53], the heuristics involve guiding the ILP solution by reducing the solution search space using binary search. With this for 7200 VNF instances, the time taken is 30 min. In [23], MILP based algorithm takes 500 s for 3000 VNFs. We have shown above that with our framework we are able to place up to 3000 VNF instances in less than 60 s. It needs to be appreciated that the results are not exactly comparable because of different experimental environments, but do give a sense of improvement with predictive algorithms.

The acceptance rate of the heuristic is an important parameter that often gets ignored. In the ongoing operations, whether we are looking at new placements or reconfiguration or migration of existing chains, it is important for the placement engine to be able to place SFCs every time a request is made subject to resources being available. If a large number of requests cannot be placed despite adequate capacities being available, then the acceptance rate is low, and we do not have a good algorithm. Failure to place SFCs would mean the loss of business for cloud service providers and may affect the requesting carriers revenue. For a medium-sized placement request, viz. 100 SFCs or 500 functions, the acceptance rate with our algorithm turns out to be 100% (Fig. 20).

As the number of service chains increases, the acceptance rate may fall because of a lack of capacity to place the complete service chains. When corrected for capacity, the acceptance rate for our algorithm remains above 98% up to the tested configuration of 500 SFCs or 2500 VNFs.

We compare this with the real-time placement presented in [54]. The authors propose an ILP model to provide an optimal solution for placement and chaining VNFs based on minimizing the resources allocation and the deployment (mapping) delay while meeting the real-time condition. They also propose a heuristic solution named Degree Based Heuristic (DBH) to minimize the end-to-end delay and resources allocation cost. A comparison of successful requests is given in Fig. 21.

The authors in [23] claim that with 500 VNFs, the acceptance rate is 85%. In comparison, for our solution, the acceptance rate is 100% for up to 100 SFCs or 500 VNFs. Above this, the acceptance rate drops to 98% for up to 2500 VNFs.

7. Summary and future work

Innovative strategies are required to extract carrier-grade performance from SFCs that use resources from multiple clouds. Our strategy consists of techniques based on a predictive approach to performance optimization. Complex performance indicators, like end-to-end latency of a service chain at activation time, depend on far too many deterministic and probabilistic factors, to be modeled accurately by deterministic techniques. We have shown that a carefully designed predictive approach combined with heuristics to select low-latency clouds can help us in keeping the performance consistent with the SLA and costs within the carrier's budget. To make latency predictions more accurate, we have worked with time-based windows and an incremental update of the models used for prediction. Making use of the predicted latencies is an iteratively convergent randomized search heuristic used to select low latency clouds for successive placement of VNFs. Not only the proposed strategy produces results with low error, but it also executes fast so that the results can be used to take corrective actions. A comprehensive empirical evaluation has been carried out and reported in this paper. The proposed P-ART framework has been built from all the techniques that have been described in this paper.

A number of research directions are foreseen in this project. When enough resources are not available, carriers may accept under-dimensioned service chains. The service has to be functional, even though not meeting the performance criteria. Another important issue to be worked upon is the security aspect of VNSs in the multi-cloud environment.

Acknowledgments

This publication was made possible by NPRP grant #8-634-1-131 from the Qatar National Research Fund (a member of Qatar Foundation), National Science Foundation, USA CNS-1718929 and National Science Foundation, USA CNS-1547380. The findings achieved herein are solely the responsibility of the author[s].

This paper is a revised and significantly expanded version of a paper entitled 'COLAP: A Predictive Framework for Service Function Chain Placement in a Multi-cloud Environment' presented at the 7th IEEE CCWC in Las Vegas in January 2017. The paper won the 'Best Paper' award in the Cloud Computing track at the conference. The similarity between the conference and this paper is about 40%.

References

[1] Network Functions Virtualization (NFV); Virtualised Network Function; Specification of the Classification of Cloud Native VNF Implementations (work-in-progress), ETSI GS NVF-EVE 011 V009, 2018.

[2] Martins M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, Clickos and the art of network function virtualization, in: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, 2014, pp. 459–473.

[3] F. Lopez-Pires, B. Baran, Virtual machine placement literature review, Tech. Rep., Polytechnic School, National University of Asuncion, 2015, [Online]. Available: http://arxiv.org/abs/1506.01509.

[4] C.J. Bernardas, A. Rahman, J.C. Zunjia, L.M. Contreras, P. Aranda, P. Lynch, Network Virtualization Research Challenges, IETF internet draft, 2018.

[5] Series G: Transmission Systems And Media, Digital Systems And Networks, The E-model: a computational model for use in transmission planning, Recommendation ITU-T G107, 2015.

[6] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: Challenges and opportunities for innovations, IEEE Commun. Mag. 53 (2) (2015) 90–97.

[7] F. Callegati, W. Cerroni, C. Contoli, G. Santandrea, Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case, in: IEEE 3rd International Conference on Cloud Networking (CloudNet), 2014, pp. 132–137.

[8] Kangkang Li, Huanyang Zheng, Jie Wu, Migration-based virtual machine placement in cloud systems, IEEE Cloudnet (2013) 83–90.

[9] R. Ricci, E. Eide, The CloudLab Team, Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications, Usenix Login 39 (6) (2014) 36–37.

[10] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare, C. Metz, COLAP: A predictive framework for service function chain placement in a multi-cloud environment, in: 7th IEEE Annual Computing and Communication Workshop and Conference, 2017, pp. 1–9.
[11] Network Functions Virtualisation (NFV); Accountability; Report on Quality Accountability Framework, ETSI GS NFV-REL 005 V111, 2016.
[12] NFV Use cases, ETSI GS NFV 0001, 2013.
[13] P. Quinn, T. Nadeau, Problem Statement for Service Function Chaining, IETF RFC 7498, 2015.
[14] D. Bhamare, R. Jain, M. Samaka, A. Erbad, A survey on service function chaining, J. Netw. Comput. Appl. (2016) 138–155.
[15] R. Yu, G. Xue, V.T. Kilari, Xiang Zhang, Network function virtualization in the multi-tenant cloud, IEEE Netw. (2015) 42–47.
[16] D.B. Oljira, K.-J. Grinnemo, J. Taheri, A. Brunstrom, A model for QoS-Aware VNF placement and provisioning, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1–7.
[17] R. Gouareb, V. Friderikos, A.H. Aghvami, Delay sensitive virtual network function placement and routing, in: 25th International Conference on Telecommunications, 2018, pp. 394–398.
[18] D. Li, P. Hong, K. Xue, J. Pei, Virtual network function placement considering resource optimization and SFC requests in cloud datacenter, IEEE Trans. Parallel Distrib. Syst. 29 (7) (2018) 1664–1677.
[19] Dynamic, Latency-Optimal vNF Placement at the Network Edge, in: IEEE INFOCOM-IEEE Conference on Computer Communications, 2018, pp. 693–701.
[20] A. Leivadeas, M. Falkner, I. Lambadaris, M. Ibnkahla, G. Kesidi, Balancing delay and cost in virtual network function placement and chaining, in: IEEE NetSoft 2018 - International Workshop on Smart network Technologies and Edge computing for the Tactile Internet (STET), 2018, pp. 433–440.
[21] S. Ahvar, H.P. Phyu, S.M. Buddhacharya, E. Ahvar, N. Crespi, R. Glitho, CCVP: Cost-efficient centrality-based VNF placement and chaining algorithm for network service provisioning, in: Proc. IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–9.
[22] Z. Xu, X. Zhang, S. Yu, Ji. Zhang, Energy-efficient virtual network function placement in telecom networks, in: IEEE International Conference on Communications (ICC), 2018, pp. 1–7.
[23] T.W. Kuo, B.H. Liou, K.C.J. Lin, M.-J. Tsai, Member, Deploying chains of virtual network functions: On the relation between link and server usage, IEEE/ACM Trans. Netw. (2016) 1562–1576.
[24] M. Dieye, S. Ahvar, J. Sahoo, E. Ahvar, R. Glitho, H. Elbiaze, N. Crespi, CPVNF: Cost-Efficient proactive VNF placement and chaining for value-added services in content delivery networks, IEEE Trans. Netw. Serv. Manag. 15 (2) (2018) 774–786.
[25] L. Zhao, J. Liu, Optimal placement of virtual machines for supporting multiple applications in mobile edge networks, IEEE Trans. Veh. Technol. (2018) 6533–6545.
[26] L. Askari, A. Hmaity, F. Musumeci, M. Tornatore, Virtual-network-function placement for dynamic service chaining in metro-area networks, in: International Conference on Optical Network Design and Modeling (ONDM), 2018, pp. 136–141.
[27] G. Nychis, D.R. Licata, The impact of background network traffic on foreground network traffic, in: The Proceeding of the IEEE Global Telecommunications Conference GLOBECOM, 2001, pp. 1–16.
[28] Amazon Opworks, https://aws.amazon.com/opsworks/, 2019, last accessed 2 January 2019.
[29] S. Clayman, E. Maini, A. Galis, A. Manzalini, N. Mazzocca, The dynamic placement of virtual network functions, in: IEEE Network Operations and Management Symposium, NOMS, 2014, pp. 1–9.
[30] A.J. Smola, B. Scholkopf, A tutorial on support vector regression, Stat. Comput. (2004) 199–222.
[31] The CAIDA anonymized internet Traces 2016 Dataset, Center for Applied Internet Data Analysis, http://www.caida.org/data/passive/passive_2016, last modified June 2018, last accessed 17 December 2018.
[32] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, B. Pfahringer, Efficient online evaluation of big data stream classifiers, in: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, 2015, pp. 59–68.
[33] Y. Sun, Z. Wang, H. Liu, Chao Du, Jidong Yuan, Online ensemble using adaptive windowing for data streams with concept drift, Int. J. Distrib. Sens. Netw. (2016) 1–9.
[34] L.M. Seversky, S. Davis, On time-series topological data analysis: New data and opportunities, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2016, pp. 1014–1022.
[35] D.P. Helmbold, P.M. Long, Tracking drifting concepts by minimizing disagreements, Mach. Learn. 14 (1) (1994) 27–45.
[36] R. Klinkenberg, Th. Joachims, Detecting concept drift with support vector machines, in: Proceedings of the 17th International Conference on Machine Learning (ICML), 2000, pp. 487–494.
[37] R.W. Becker, G.V. Lago, A global optimization Algorithm, in: Proceedings of the 8th Allerton Conf. Circuits Systems Theory, 1970.
[38] F.J. Solis, J b. Wets, Minimization by random search techniques, in: Mathematics Of Operations Research, 1998.
[39] T. Weis, Global Optimization Algorithms – Theory and Application – e-book, http://www.it-weise.de/projects/book.pdf, 2009, last accessed 10 December 2018.
[40] Zelda B. Zabinsky, Random Search Algorithms, Wiley Encyclopedia of Operations Research and Management Science, 2009.
[41] A. Zhigljavsky, Stochastic Global Optimization, Springer book ed., School of Mathematics, Cardiff University, Cardiff, UK, 2008.
[42] C.J.P. Bélisle, Convergence theorems for a class of simulated annealing algorithms on rd, J. Appl. Probab. 29 (1992) 885–895.
[43] W.A. Rankothge, J. Me, F. Le, A. Russo, J. Lobo, Towards making network function virtualization a cloud computing service, in: IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 89–97.
[44] V. Gupta, S. Dharmaraja, V. Arunachalam, Stochastic modeling for delay analysis of a VoIP network, Ann. Oper. Res. (2015) 171–180.
[45] A. Akella, Experimenting with next-generation cloud architectures using CloudLab, IEEE Internet Comput. (2015) 77–81.
[46] M.J. Kearns, The Computational Complexity of Machine Learning, MIT Press, 1990, 176 pages.
[47] S. Raschka, Model evaluation, model selection, and algorithm selection in machine learning, 2018, arXiv:1811.12808v2 [cs.LG].
[48] E. Frank, M.A. Hall, I.H. Witten, The weka workbench, in: Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques, fourth ed., Morgan Kaufmann, 2016.
[49] M.C. Luizelli, L.R. Bays, L.S. Buriol M. P. Barcellos, L.P. Gaspary, Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions, IFIP, 2015.
[50] L. Wang, et al., Joint optimization of service function chaining and resource allocation, IEEE Access (2018).
[51] M.T. Beck, J.F. Botero, Coordinated allocation of service function chains, in: Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2015, 1–6.
[52] J. Liu, Y. Li, Y. Zhang, L. Su, D. Jin, Improve service chaining performance with optimized middlebox placement, IEEE Trans. Serv. (2017).
[53] S.A. Ajila, A.A. Bankole, Cloud client prediction models using machine learning techniques, in: IEEE 37th Annual Computer Software and Applications Conference, 2013.
[54] A. Alleg, R. Kouah, T. Ahmed Moussaoui, Virtual Network Functions Placement and Chaining for Real-Time Applications, 2017, pp. 1–6.

Further reading

[1] T.O. Ayodele, Types of Machine Learning Algorithms, University of Portsmouth, UK published by Intech, 2010.

Lav Gupta is a senior member of IEEE. He received BS and MS degrees from the Indian Institute of Technology (IIT) in 1978 and 1980, respectively. He is currently a doctoral candidate in Computer Science and Engineering at Washington University in St. Louis, Missouri, USA. He has worked for about fifteen years in the area of telecommunications planning, deployment, and regulation. He has also worked as a senior faculty of Computer Science and Access Network Planning in India and the UAE for a total of about fifteen years. He is the author of one book, ten first author papers and has been a speaker at many international seminars. His current research areas are virtual network services, multi-cloud systems, fault and performance management in cloud-based Network Function Virtualization and application of AI in the management of virtual network services over clouds

Raj Jain is currently the Barbara J. and Jerome R. Cox, Jr., Professor of Computer Science and Engineering at Washington University in St. Louis. Dr. Jain is a Life Fellow of IEEE, a Fellow of ACM, a Fellow of AAAS, a recipient of 2018 James B. Eads Award from St. Louis Academy of Science, 2017 ACM SIGCOMM Life-Time Achievement Award, 2015 A.A. Michelson Award from Computer Measurement Group and ranks among the Most Cited Authors in Computer Science. Previously, he was one of the Co-founders of Nayna Networks, Inc - a next-generation telecommunications systems company in San Jose, CA. He was a Senior Consulting Engineer at Digital Equipment Corporation in Littleton, Mass and then a professor of Computer and Information Sciences at Ohio State University in Columbus, Ohio.

Aiman Erbad is an Assistant Professor in the Department of Computer Science and Engineering (CSE) at Qatar University. Dr. Erbad obtained a Ph.D. in Computer Science from the University of British Columbia (Canada) in 2012, a Master of Computer Science in Embedded Systems and Robotics from the University of Essex (UK), and a Bachelor of Science in Computer Engineering from the University of Washington (USA). Since September 2016, Dr. Erbad is the Director of Research Support, responsible for all research grants and contracts. Before that, he was the Coordinator of the Computer Engineering program and the Chair of the Curriculum and Quality Assurance committee leading ABET accreditation and curriculum enhancement efforts at the CSE department. Dr. Erbad is a recipient of the Platinum Education Excellence Award (Ph.D. Category) in 2013. His research interests span cloud computing, multimedia systems and networking, and security. His research received funding from the Qatar National

Research Fund, and his research is published in reputed international conferences and journals. Dr. Erbad is a member of various University committees (Policy, Ranking, Institutional Effective, Intellectual Property, Appeal, and Reinstatement) and the Chair of the University Research Support Committee. He serves as an Editor in the European Alliance for Innovation (EAI) Endorsed Transactions on Collaborative Computing, and as a technical program committee member in various IEEE and ACM international conferences. Dr. Erbad acts as an expert in information technology strategy and research techniques for various national entities.

Deval Bhamare is currently working as a post-doctorate at Karlstad University, Sweden. He has earned his Ph.D. from IITB-Monash Research Academy, a joint venture between Indian Institute of Technology, Bombay, India and Monash University, Melbourne, Australia. He has earned his Master of Science in Computers from the University of Southern California, LA and Bachelor of Engineering from VJTI, Mumbai, India. His areas of research include Network Optimization, Middleware Architecture for Cloud-Based Services, Cloud Security and Software Defined Networks.