ClusPR: Balancing Multiple Objectives at Scale for NFV Resource Allocation

Y. T. Woldeyohannes, Ali Mohammadkhan, K. K. Ramakrishnan *Fellow, IEEE,* and Yuming Jiang, *Senior Member, IEEE*

***Abstract*—Network Function Virtualization (NFV) implements network middleboxes in software, enabling them to be more flexible and dynamic. NFV resource allocation methods can exploit the capabilities of virtualization to dynamically instantiate network functions (NFs) to adapt to traffic demand and network conditions. Deploying NFs requires decisions for NF placement, and routing of flows through these NFs in accordance with the sequence of NFs required to process each flow. The challenges in developing an NFV resource allocation scheme include the need to manage the dependency between flow-level (routing) and network-level (placement) decisions and to efficiently utilize resources that may be distributed network-wide, while fulfilling the performance requirements of flows.**

We propose a scalable resource allocation scheme, called ClusPR, that addresses these challenges. By elegantly capturing the dependency between flow routing and NF placement, ClusPR strikes a balance between multiple objectives including *minimizing path stretch*, *balancing the load among NF instances*, while *maximizing the total network utilization* by accommodating the maximum number of flows possible. ClusPR addresses the offline problem of NFV resource allocation. To address the online problem of dynamically placing and routing flows upon their arrival, we propose iClusPR. iClusPR is an online algorithm that performs dynamic scaling by adjusting the number of NF instances based on the traffic demand and the network state.

Our experiments show that ClusPR achieves the near-optimal solution for a practical large-sized network in reasonable time. Compared to the state-of-the-art approaches, ClusPR decreases the average normalized delay by a factor of $1.2 - 1.6\times$ and the worst-case delay by more than $10\times$, with the same or slightly better network utilization and balances the load among NF instances. Furthermore, the performance of iClusPR, the online version, is comparable to the offline ClusPR algorithm.

***Index Terms*—NFV, orchestration, placement, flow routing, multi-objective, load balancing, clustering.**

I. Introduction

Middleboxes such as firewalls, VPN gateways, proxies, intrusion detection and prevention systems play a central role in today's Internet by providing network resident functionality that examines and potentially modifies the end-to-end traffic flow [2]. Implementing middleboxes is gradually migrating to software platforms, providing additional flexibility and extensibility for the capabilities of the network compared to purpose-built hardware appliances. Evolving the network's capabilities can thus involve lower capital expenditures as the software can run on commercial off-the-shelf (COTS) hardware. Network Function Virtualization (NFV) decouples the software of network functions from the physical machine and runs it on virtual machines, or more recently on "containers" [3]. This also brings greater flexibility in resource management as instances of the network functions (NFs) can be created dynamically, and the capacity for a particular function can be scaled up or down depending on the traffic demand. Sequences of NFs are common, and the overall service provided by the network by such sequences of NFs is termed "service function chaining" [4]. Using the capability of Software Defined Networks (SDN) to perform flow specific routing, we can route the flows requiring the service chain functionality through the NF service instances.

A. The Challenge

Resource allocation of NF service chains continues to be a challenge because of the complexity of making network-level as well as flow-level decisions holistically, since each impacts the other. An NFV resource allocation (NFV-RA) mechanism has determine at the network-level the number of NF instances to instantiate across all the nodes in the network to process all admitted flows. Further, the algorithm needs to determine the placement of the NFs or the physical machines that should host the NFs. The flow-level decision making includes the assignment of NF instances for the service chain of the flow and the determination of the route for each flow. However, the network-level decision making has to be coupled with the decision making at the flow-level, because placement decisions resulting in NFs placed on network nodes not along the shortest path will result in "path stretch", contributing to increased latency for the flow. This interdependence between network-level (placement, instantiating the requisite number of NFs) and flow-level (instance assignment, routing) decisions makes the NFV-RA problem new and challenging, warranting the recent attention it has received in the research community.

In addition, network resources such as bandwidth of links, memory and processing capacities of nodes are limited. These resource limitations call for efficient utilization of the available resources, which could be achieved for example by minimizing the number of NF instances, balancing the load among the NF instances and/or minimizing the number of distinct resources used. However, these efficient resource utilization objectives could sometimes be in conflict with the objective of fulfilling the performance requirement of flows. Thus, one of the challenges in developing an NFV resource allocation scheme is to

Y. T. Woldeyohannes and Yuming Jiang are with the Department of Information Security and Communication Technology, Norwegian University of Science and Technology, NTNU, Norway

Ali Mohammadkhan and K. K. Ramakrishnan are with the Department of Computer Science and Engineering, University of California, Riverside, California, USA.

Preliminary version of this article appeared in ICIN, 2018 conference [1].

find a balance across multiple objectives, involving minimizing path stretch, maximizing the number of flows accommodated, and balancing the load among NF instances, while considering the resource constraints in the network.

The end-to-end latency for flows can potentially decrease if NF instances are placed closer to users instead of routing the network traffic to a limited number of centralized data-centers [5]. However, most of the work in the literature focus only on the placement of NFs in the Cloud e.g., Stratos [6] and [7] or at central offices (COs), as in E2 [8]. In this paper, we *develop a scalable resource allocation scheme for the joint NF placement and flow routing problem in a geo-distributed, general, network. The novelty of our approach comes from considering multiple objectives at the same time (minimizing path stretch, maximizing the total utilization of the network and balancing the load among NF instances)*. Our work furthers the state of the art by capturing the inter-dependency between flow-level and network-level decisions in a scalable manner and striking a balance across multiple objectives. While existing works typically restrict the flow to the shortest-path avoiding path stretch, and disregarding its effect on network utilization [9], [10]. Others maximize network utilization at the expense of increased path stretch [11], [12].

B. Our Contributions

Our first contribution is that we model the joint NF placement *and* flow routing problem as a multi-objective integer linear programming (ILP) problem. The model is able to allocate NF instances and find the end-to-end route of flows while maintaining the precedence constraint among NFs of the service chain. We solve the ILP using a conventional solver (CPLEX) for a reasonable scale problem with realistic parameters. Although the ILP formulation gives optimal results, the run time of the algorithm increases exponentially with the size of the network and/or the number of flows. This is because the joint NF placement and flow routing problem is *NP-hard*, encompassing the two NP-hard problems (placement and flow routing) [13]. The results obtained from solving the ILP model provide us with valuable insights to develop a heuristic solution capable of solving the NFV-RA problem at a larger scale in reasonable time.

Secondly, taking as input the insights, we develop a heuristic-based NFV-RA scheme called ClusPR. ClusPR strikes a balance between multiple objectives including *minimizing the path stretch* experienced by flows, *maximizing the total utilization of the network* (the number of flows admitted) and *balancing the load among NF instances*. ClusPR utilizes a divide-and-conquer approach, decomposing the NFV-RA problem into two sub-problems: for NF placement and flow routing. To capture the dependency between the network-level (NF placement) and flow-level (flow routing) decisions, ClusPR adapts a novel hierarchical architecture in which first flows are grouped based on their path proximity. Then, the route information of flows is extracted and used as input when making NF placement decisions.The features of ClusPR are:

- **Network:** ClusPR can be used in general settings where NF instances may be hosted not only in the cloud but also on the edge computing nodes.
- **Path stretch vs utilization:** Path stretch can be avoided if NF instances that are needed by a flow are placed on its (shortest) path as done in [9]. This can however lead to low utilization of the network. ClusPR strikes a balance between minimizing path stretch and maximizing the network utilization.
- **Load balancing vs path stretch:** Studies have demonstrated that NF overload is a common cause of failures and therefore it is important to balance the load across NFs [2], [14]. However, such load balancing decisions might lead to increased delay and thus violating Service Level Agreement (SLA) for flows. Our proposed flow routing algorithm balances the load among NF instances while taking into account the delay requirement of flows without redirect existing flows.

Finally, we propose an online algorithm iClusPR, which dynamically scales the number of NF instances depending on the traffic demand and network state. iClusPR makes resource allocation decisions on a time slot basis.

- **Online algorithm features:** iClusPR has the same design principle as the offline algorithm, ClusPR, so it also has the aforementioned features of ClusPR.
- **Experiment:** The performance of iClusPR is analyzed through realistic experiments in which flows arrive randomly and depart after being served for a random amount of time.

II. The System Model

We consider a network of nodes and links, modeled as a directed graph, $G(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes in the network and $\mathcal{L}$ is the set of links interconnecting the nodes. A node can be in a data-center with multiple servers, a router or a commercial off-the-shelf (COTS) server along with a router. The network carries a set of flows, $\mathcal{F}$, and supports a set of NFs, denoted as $\mathcal{V}$. For each NF type, multiple instances may be instantiated on one or multiple nodes.

A node n is characterized by the number of CPU cores at the node, denoted as K_n, and memory capacity, denoted as M_n. A CPU core is dedicated to a single NF instance [15], and does not span CPU cores to avoid Non-Uniform Memory Access (NUMA) overheads. An NF instance can be hosted on any node that has enough available resources.

An NF instance $v \in \mathcal{V}$ hosted on node n is characterized by its service rate of requests, μ_n^v. Here, we assume that the instances of the same NF type at the same node have the same service rate. An instance of type v needs k^v cores and m^v amount of memory. In addition, an NF instance can process multiple flows whose service chains include the type of NF instance. We use D_n^v to denote the expected nodal delay for type v NF at node n, consisting of both processing delay and queuing delay for flows with NF type v at the node.

Each link $l \in \mathcal{L}$ is assumed to be bi-directional, and we use $l_{n'}^n$ to represent the link from node n to $n^{'}$ and C_l its expected transmission rate. Each node n has a set of outgoing

links represented by $\mathcal{L}_n^{out}$ and a set of incoming links, $\mathcal{L}_n^{in}$. The expected delay on link l, that comprises transmission and propagation delays, is written as D_l.

A flow $f \in \mathcal{F}$ is a sequence of data packets that are generated at expected rate λ_f and sent from a source to a destination node, traversing a sequence of intermediate nodes and links in the network. Each flow f has a specified service chain of NFs, denoted as $\overrightarrow{S}_f = (S_f^1, S_f^2 ..., S_f^{J_f})$, which is an *ordered* sequence of required NFs that the flow's packets must go through, where $S_f^j \in \mathcal{V}$ denotes the j^{th} NF on flow f's service chain and $J_f := |\overrightarrow{S}_f|$ is the length of the NF chain of flow f.

In addition to the sequence of NFs to be followed for the service chain, each flow f also has an end-to-end delay requirement, denoted as D_f, between the source node s_f and the destination node d_f of the flow. The end-to-end delay is composed of two types of delays: total delay on links, denoted as D_T, and total delay on nodes, denoted as D_P.

III. The NFV Resource Allocation Problem

As discussed earlier, in a network supporting NFV, resource allocation decisions should be made both at the network-level and the flow-level. For the former, an NFV resource allocation mechanism needs to decide the number of NF instances to instantiate in the network to process the flows and where or at which nodes in the network such NF instances should be placed. For the latter, the mechanism needs to decide how to route the flows to go through the NF instances according to the order of NFs in their service chains and at the same time try to meet the flows' delay requirements by limiting path stretch.

For this NFV resource allocation problem, three objectives are of interest: *(1) to maximize the number of flows admitted to the network, (2) to minimize the use of nodal processing capacities or cores, and (3) to minimize the utilization of link capacities*, where (2) and (3) are purposed to maximally leave resources for future use.

A flow is admitted to the network if and only if there are NF instances that can serve all the services in the NF service chain of the flow without violating the flow's delay requirement. Let indicator variable $I_n^v(f;j) = 1$ denoting that an NF instance v hosted at node n is used by the j^{th} service on the service chain of flow f, and indicator variable $I_l(f;j,n_j;j+1,n_{j+1}) = 1$ denoting that link l is used by flow f to route from the j^{th} to $(j+1)^{th}$ service/NF hosted at node n_j and n_{j+1} respectively. Since more than one instance of the same NF type may be hosted at the same node, we use an integer decision variable y_n^v to represent the number of type v NF instances that are hosted at node n. Note that, while y_n^v is a network-level decision variable that decides the number and placement of NF instances, $I_n^v(f;j)$ and $I_l(f;j,n_j;j+1,n_{j+1})$ are flow-level decision variables that specify which NF instances will be used by a given flow and which links will be used in routing that flow through, respectively.

The three objectives can then be respectively represented as

$$\text{maximize} \quad \sum_{\forall f \in \mathcal{F}} \left\lfloor \frac{\sum_{j=1}^{J_f} \sum_{\forall n \forall v} I_n^v(f;j)}{J_f} \right\rfloor \tag{1}$$

$$\text{minimize} \quad \sum_{\forall n \forall v} \frac{y_n^v k^v}{K_n} \tag{2}$$

$$\text{minimize} \quad \sum_{\forall l \forall f} \sum_{\forall j}^{J_f} \frac{I_l(f;j,n_j;j+1,n_{j+1})\lambda_f}{C_l} \tag{3}$$

The above three objective functions are combined into a single-objective function, using the traditional weighted sum method [16]. Since, maximizing a given function is equivalent to minimizing the negative of the function, the single-objective function is to minimize the summation of the objective functions (2), (3) and negative of (1). The objective functions are weighted equally, with unit weights. For positive weights, the optimal solution of the single-objective representation is also a Pareto optimal solution of the multi-objective problem [16].

A. Constraints

In solving the ILP problem, a number of constraints must be satisfied, which can be classified into three categories: *capacity constraints, delay constraints,* and *NF chaining constraints*. The first category is capacity constraints, which ensure that the total traffic rate on any link does not exceed the link's transmission capacity (i.e., Constraint (4)), the total number of cores allocated to NF instances at any node does not exceed the number of cores at this node (Constraint (5)) and memory capacity (Constraint (6)), and the total processing capacity required for the admitted flows flowing through any NF instance does not exceed that instance's processing capacity (Constraint (7)):

$$\sum_{\forall f} \sum_{\forall j}^{J_f} I_l(f;j,n_j;j+1,n_{j+1})\lambda_f < C_l \quad \forall l \in \mathcal{L} \tag{4}$$

$$\sum_{\forall v} y_n^v k^v < K_n \quad \forall n \in \mathcal{N} \tag{5}$$

$$\sum_{\forall v} y_n^v m^v < M_n \quad \forall n \in \mathcal{N} \tag{6}$$

$$\sum_{\forall f} \sum_{\forall j}^{J_f} I_n^v(f;j)\lambda_f < \mu_n^v \quad \forall v \in \mathcal{V}, \forall n \in \mathcal{N} \tag{7}$$

Related and hence put in this category are two constraints implied by the definition of $I_n^v(f;j)$. One is that a flow is assigned to use NF instance of type v on node n only if there is at least one instance of NF type v hosted on node n. (i.e., Constraint (8)). Another is the constraint that any NF in the service chain of any flow is served by only one such NF instance (i.e., Constraint (9)).

$$y_n^v \geq I_n^v(f;j) \quad \forall n \in \mathcal{N}, \forall v \in \mathcal{V}, \forall f \in \mathcal{F}, \forall j \in \{1 \dots J_f\} \tag{8}$$

$$\sum_{\forall n \forall v} I_n^v(f;j) = 1 \quad \forall f \in \mathcal{F}, \forall j \in \{1 \dots J_f\} \tag{9}$$

The second category are delay constraints to ensure that a flow is admitted only if its end-to-end delay requirement is met (i.e., Constraint (10)):

$$D_T + D_P < D_f \quad \forall f \in \mathcal{F} \tag{10}$$

where $D_T = \sum_{\forall l} \sum_{\forall j \forall n_j \forall n_{j+1}}^{J_f} I_l(f;j,n_j;j+1,n_{j+1}) D_l$ and $D_P = \sum_{\forall j}^{J_f} \sum_{\forall n \forall v} I_n^v(f;j) D_n^v$ are respectively the total link delay and the total nodal delay that the flow f will experience in the network if admitted.

The third category is NF chaining constraints, which are to ensure that the order of NFs of any flow is followed when it is routed from its source node, through NF instances of its service chain and finally to its destination node. As shown below, several constraints, i.e., (11) – (20), are in this category. In these constraints, variable j is used to represent the service order, i.e., $j = 1$ represents the first service, $j = 2$ the second service, and so on in an NF service chain. The source and the destination are represented by $j = 0$ and $j = J + 1$ respectively.

We start with Constraints (11) and (12) that can be regarded as the flow conservation equations for the set of nodes that host NF instances assigned to serve a flow. Specifically, Constraint (11) ensures that one of the *outgoing links of node* n_j that is running j^{th} service of flow f has to be assigned for routing flow f from its j^{th} to $(j+1)^{th}$ service order. The NF assignment decision variables $I_{n_j}^v(f;j)$ and $I_{n_{j+1}}^v(f;j+1)$ are multiplied to make sure that the constraint applies to the case where node n_j is used to serve the j^{th} service and node n_{j+1} the $(j+1)^{th}$ service of flow f. Similarly, Constraint (12) makes sure that one of the *incoming links of node* n_{j+1} that runs $(j+1)^{th}$ service of flow f has to be assigned for routing flow f from its j^{th} to $(j+1)^{th}$ service. We remark that both (11) and (12) are not linear but since they are multiples of binary variables, they can easily be substituted by a set of linear equations.

$$\forall f \in \mathcal{F}, \forall j \in \{1 \dots J_f - 1\}, \forall n_j, n_{j+1} \in \mathcal{N}:$$

$$\sum_{l \in \mathcal{L}_{n_j}^{out}} I_l(f;j,n_j;j+1,n_{j+1}) I_{n_j}^v(f;j) I_{n_{j+1}}^v(f;j+1) = 1 \tag{11}$$

$$\sum_{l \in \mathcal{L}_{n_{j+1}}^{in}} I_l(f;j,n_j;j+1,n_{j+1}) I_{n_j}^v(f;j) I_{n_{j+1}}^v(f;j+1) = 1 \tag{12}$$

Constraints (13) and (14) are used to guarantee that no more than one link outgoing from or incoming to a node are assigned to a given service order of a flow respectively:

$$\forall n \in \mathcal{N}, \forall f \in \mathcal{F}, \forall j \in \{0, ..J_f\}:$$

$$\sum_{l \in \mathcal{L}_n^{out}} \sum_{n_j, n_{j+1} \in \mathcal{N}} I_l(f;j,n_j;j+1,n_{j+1}) \leq 1 \tag{13}$$

$$\sum_{l \in \mathcal{L}_n^{in}} \sum_{n_j, n_{j+1} \in \mathcal{N}} I_l(f;j,n_j;j+1,n_{j+1}) \leq 1 \tag{14}$$

Constraint (15) is a flow conservation constraint of the *intermediate nodes*, which are nodes that do not host NF instances that are assigned to the flow but still should route the flow in a given order of the route. It makes sure that, if one of the incoming links of a node is assigned for a given order of the route then one of the outgoing links of the same node has to be assigned to the same order.

$$\sum_{l \in \mathcal{L}_n^{in}} I_l(f;j,n_j;j+1,n_{j+1}) - \sum_{l \in \mathcal{L}_n^{out}} I_l(f;j,n_j;j+1,n_{j+1}) = 0, \forall f \in \mathcal{F}, \forall j \in \{0, ..J_f\}, \forall n \in \mathcal{N}/n \neq n_j, n_{j+1}, s_f, d_f \tag{15}$$

Constraints (16) and (17) are *source node* flow conservation constraints. Constraint (16) ensures that one of the outgoing links of the source node of flow f is used to route from the 0^{th} to 1^{st} service of the service chain. As defined earlier, the source node represents the 0^{th} service. That is, node n_0 is equivalent to s_f, the source node of flow f. In addition, Constraint (17) is used to assign one of the incoming links of a node (n_1), serving the 1^{st} service of the service chain, to the 1^{st} service order of the service chain.

$$n_0 = s_f, \forall n_1 \in \mathcal{N}, \forall f \in \mathcal{F}:$$

$$\sum_{l \in \mathcal{L}_{n_0}^{out}} I_l(f;0,n_0;1,n_1) = I_{n_1}^v(f;1) \tag{16}$$

$$\sum_{l \in \mathcal{L}_{n_1}^{in}} I_l(f;0,n_0;1,n_1) = I_{n_1}^v(f;1) \tag{17}$$

Constraints (18) and (19) are flow conservation constraints for the (*destination node*). Constraint (18) makes sure that one of the incoming links of the destination node is assigned to route from the node hosting the last NF of the service chain (n_J) to the destination node (d_f) that is also represented as the $J+1$ service, as defined earlier. In addition, Constraint (19) assigns one of the outgoing links of the node serving the last service to the $(J+1)^{th}$ service order.

$$n_{J+1} = d_f, \forall n_J \in \mathcal{N}, \forall f \in \mathcal{F}:$$

$$\sum_{l \in \mathcal{L}_{n_{J+1}}^{in}} I_l(f;J,n_J;J+1,n_{J+1}) = I_{n_J}^v(f;J) \tag{18}$$

$$\sum_{l \in \mathcal{L}_{n_J}^{out}} I_l(f;J,n_J;J+1,n_{J+1}) = I_{n_J}^v(f;J) \tag{19}$$

Finally, Constraint (20) is used to avoid loops: If link $l_{n'}^n$ is assigned for a given service order then link $l_n^{n'}$ should not be assigned to the same service order.

$$I_{l_{n'}^n}(f;j,n_j;j+1,n_{j+1}) + I_{l_n^{n'}}(f;j,n_j;j+1,n_{j+1}) \leq 1, \forall l \in \mathcal{L}, \forall f \in \mathcal{F}, \forall j \in \{0 \dots J_f\}, \forall n_j, n_{j+1} \in \mathcal{N} \tag{20}$$

B. Observations from Solving the ILP Problem

We implemented our ILP model and used CPLEX to arrive at the optimal placement on a number of small size networks. We realize that the results from these small scale experiments are not generalizable, but the results provide us with valuable guidance to develop a heuristic solution that can solve the placement and routing problem at a much larger scale. Based on solving the ILP model for a number of scenarios (service chains varying in length, nodes having varying number of cores, and NFs needing multiple, but different numbers of cores), we make the following observations.

1) **Observation 1 (O1)**: The nodes chosen for NF placement are usually on the shortest path of at least one of the flows. If a node that is on the shortest path of all the flows has enough capacity to host all the NFs required for those flows, the optimal solution places all the NFs on that node.
2) **Observation 2 (O2)**: In the optimal solution, different NF types are usually placed on a node rather than multiple instances of the same NF type. If various types of NFs are placed on a node, a flow will be able to get all the services of its chain in one node with a high probability. This is especially true for flows with shorter service chains.
3) **Observation 3 (O3)**: The total number of flows that can be accommodated in the network is a function of many factors, including the amount of network resources available, the NFs' service rate and flow types. Prioritizing flows that require popular services (at least one) increases the total number of flows that can be accommodated in the network. This is because, if instances that are needed by very few flows are instantiated a priori, those instances will occupy resources but will be under utilized, thus not accommodating popular NFs that could have been utilized more.
4) **Observation 4 (O4)**: To satisfy the performance requirements (e.g., stringent delay) of some flows, more than the minimum number of NF instances needed may be instantiated.

IV. ClusPR: A Heuristic Approach for NFV Resource Allocation

Based on the observations summarized in Sec. III-B, we develop ClusPR, a scalable NFV resource allocation algorithm. ClusPR consists of three phases: Initialization/**Clus**tering, **P**lacement, and **R**outing phase. Fig. 1 shows the overall design of ClusPR.

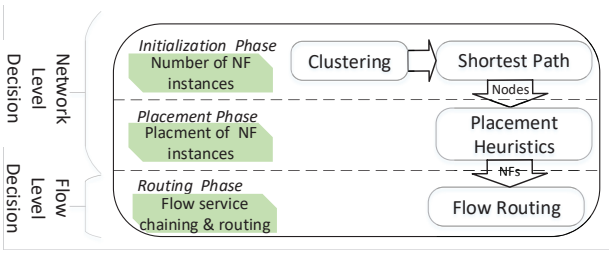


Fig. 1: ClusPR Resource Allocation Phases

Network resources (such as CPU cores, memory, bandwidth) should be used efficiently so as to maximize the total utilization of the network (i.e., the total number of flows admitted to the network with the minimum resources). On the other hand, path stretch of flows needs to be minimized, as increased latency could lead to violation of a flow's SLA. However, these objectives could be conflicting. For example, path stretch can be avoided if flows are served by NF instances placed on their shortest path as in [9], [10]. However, relatively few flows share a specific end-to-end path and those that do might have different service chain requirements. So this could lead to under-utilization of the network. For increasing the utilization of the network, an NF instance should serve a larger number of flows.

Given that a number of flows may need to share the same NF instance, that NF may not be on the shortest path of all of the flows it serves. Thus, a significant number of flows may have to deviate from their shortest path to be served by the NFs instantiated. However, this deviation needs to be constrained so as not to violate the delay requirement of flows. To strike a balance between minimizing path stretch and maximizing network utilization, ClusPR groups flows based on their path information. That is, flows whose paths are in close proximity to each other will be grouped together. Then flows within a group will share NF instances. The intuition is that since flows within a group have paths that are 'close' to each other, they can share NF instances with zero or minimal path stretch.

After grouping the flows, their path information is extracted by identifying the nodes that are on the shortest path of each group of flows. Then, NF instances are placed and assigned to flows, and flows are routed end-to-end. In Summary, ClusPR's design principle is to first *(1)group flows based on their path proximity, (2) then extract the flows' path information, (3) place instances considering the path information and (4) finally, assign NF instances and route flows* taking into account their performance requirement and the utilization level of the instances.

A. *Initialization Phase*

*1) **Clustering**:* The purpose of the clustering module is to find a set/group of flows whose paths are in close proximity to each other. To group the flows, ClusPR clusters access nodes (e.g., first/last hop routers to sources and destinations in a wide-area network). Access nodes are clustered based on their proximity to each other, i.e., access nodes that are close to each other are clustered together. *Because of the topological proximity between clustered access nodes, flows originating from access nodes of a cluster and going to access nodes in another cluster or vice versa will have their paths close to each other*. Thus, these flows can also potentially share NF instances with minimal path stretch. Inspired by this insight, ClusPR groups flows into intra-cluster (i.e., flows whose source and destination nodes are in the same access cluster) and inter-cluster flows (i.e., flows whose source and destination nodes belong to two different clusters) and performs the placement of NF instances for each of these groups of flows independently.

To cluster the access nodes, a network clustering algorithm, Kruskal's algorithm [17], which is a minimum spanning tree (MST) based clustering algorithm [18], is used. Given a network graph $G(\mathcal{N}, \mathcal{L})$, the algorithm first organizes all nodes/vertices of the graph in disjoint sets with a set containing one node/vertex. The edges of the graph are sorted and listed in ascending order, based on their weight, which is equal to the delay on the edge. Then, an edge that is on top of the sorted list of edges is considered. If the vertices of the edge belong to two different disjoint sets, the sets will be merged into a single set. In order to find an MST, this step is repeated until all the edges in the sorted list are visited.

Given the number of clusters k, an MST based clustering algorithm can be used to find the k clusters by sorting edges and merging sets until the heaviest $k-1$ edges are left in the sorted edge list [19].

*a) **Optimal Number of Clusters**:* Kruskal's algorithm can be used to find clusters given that the number of clusters is known. But finding the optimal number of clusters apriori might be challenging especially if the network size is large. To solve this problem, cluster validation techniques can be used to automatically find the optimal number of clusters [20]. One of the classical cluster validation techniques is the Dunn index [21]. The Dunn index is useful for finding dense and well-separated clusters. It is defined as the ratio between the minimum inter-cluster distance to the maximum intra-cluster distance.

$$\mathcal{D}(k) = \min_{i,j \in \{1 \ldots k\}, i \neq j} \left\{ \frac{\delta_{i,j}}{\max_{1 \leq l \leq k} \triangle_l} \right\} \tag{21}$$

Here $\delta_{i,j}$ is the inter-cluster distance between clusters i and j, defined as the minimum distance between a pair of nodes across clusters i and j, i.e., $\min_{x_i \in C_i}\{\min_{x_j \in C_j} d(x_i, x_j)\}$, C_i is the set of nodes in cluster i. The distance metric, $d(x_i, x_j)$, is the edge cost or the shortest path cost between the nodes. The intra-cluster distance of a given cluster l, $\triangle_l$, is defined as the maximum distance between a pair of nodes within that cluster i.e., $\max_{x_l \in C_l}\{\max_{x_k \in C_l} d(x_l, x_k)\}$.

The Dunn index will be maximum when the minimum inter-cluster distance is large and the maximum intra-cluster distance is small. Larger values of the Dunn index corresponds to good clusters. Therefore, *the number of clusters (k) that maximizes the Dunn index is taken as the optimal number of clusters* [22].

Algorithm 1 ClusPR: Clustering Phase

1: $G(\mathcal{N}, \mathcal{L})$: where n , $n^{'} \in \mathcal{N}$
2: sort the links/edges $l \in \mathcal{L}$ in ascending order
3: $k \leftarrow N$ number of disjoint sets(clusters)
4: $k_{opt} \leftarrow k$
5: **for** $l : \{n, n^{'}\} \in \mathcal{L}$ **do**
6: **if** n and $n^{'}$ are in disjoint sets **then**
7: merge sets' of n and $n^{'}$
8: $k \leftarrow k - 1$
9: **if** $\mathcal{D}(k) > \mathcal{D}(k_{opt})$ **then**
10: $k_{opt} \leftarrow k$
11: return k_{opt}
12: Kruscal's Algorithm(k_{opt})

Finding the Optimal Number of Clusters: To find the optimal number of clusters, Kruskal's algorithm can be run multiple times for different numbers of clusters, and the number of clusters that maximizes the Dunn index value will be considered the optimal. However, this approach can be computationally cumbersome as it requires running the clustering algorithm multiple times. To efficiently calculate the optimal number of clusters, we propose an approach that exploits the structure of Kruskal's clustering algorithm.

Kruskal's clustering algorithm is a hierarchical algorithm in which the number of clusters decreases at each step until the target number of clusters is reached. In the beginning, the number of clusters is equal to the number of nodes in the network (N). The number of clusters decreases step by step from N until the target number of clusters is reached. The Dunn index can be calculated in the middle of the MST based clustering algorithm execution.

For example, in the beginning the Dunn index can be calculated for N clusters, before two clusters are merged and then the number of clusters decreases to N-1. The Dunn index is then calculated for N-1 clusters before two sets are merged and the cluster number becomes N-2 and so on. Algorithm 1 shows the pseudo code for efficiently calculating the Dunn index inside the Kruskal's clustering algorithm. At line 10, the Dunn index for the current number of clusters k is calculated using equation (21). If it is more than the Dunn index of the temporarily optimal number of clusters (k_{opt}), k will take the place of k_{opt}. After getting the optimal number of clusters that has the maximum value of the Dunn index, Kruskal's clustering algorithm will finally be run for the optimal number of clusters (line 13).

Number of NF Instances: Latency-sensitive flows have less tolerance for path stretch. Thus, as noted in **O4**, a larger number of NF instances might have to be instantiated to satisfy the delay requirement for this type of flows. In this section, we propose two ways for deciding the number of NF instances to be created in the network. In the first approach, the minimum number of instances required to serve all the flows in the set ($\mathcal{F}$) is calculated first, then these instances are divided among the groups of flows. In the second approach, the minimum number of instances needed to serve each group of flows is calculated first, then the total number of instances instantiated will be a summation of the number of instances created for the groups.

First approach: this approach creates the minimum number of instances needed to serve all the flows in the set $\mathcal{F}$.

Theorem 1. *Given a set of flows ($\mathcal{F}$), each flow having an average arrival rate of λ, out of which F^v number of flows require the v type NF. The minimum number of NF instances of type $v \in \mathcal{V}$ ($\mathcal{I}^v_{min}$), with service rate μ^v, required to serve the set of flows is equal to $\mathcal{I}^v_{min} = \left\lceil \frac{F^v \lambda}{\mu^v} \right\rceil$.*

Proof: refer to Appendix A.

The minimum number of instances of each type of NF ($v \in \mathcal{V}$) needed to serve the flows in $\mathcal{F}$ is calculated using Theorem 1. The calculated number of instances will then be divided among the groups of flows.

- *For large numbers of flows*: If the number of flows in $\mathcal{F}$ is large, the minimum number of NFs calculated can be proportionally divided among the groups of flows. Let $\mathcal{G}$ represent the set of groups of flows. For each group $g \in \mathcal{G}$, the number of NF instances of type v instantiated ($\mathcal{I}^v_g$) is allocated in proportion to the number of flows in the group that require NF type v (F^v_g) to the total number of flows that need v, F^v. That is $\mathcal{I}^v_g = \frac{\mathcal{I}^v_{min} F^v_g}{F^v}$. Since this fraction might not always be an integer number, two conditions are used for assigning positive integer values to $\mathcal{I}^v_g$. If $\frac{\mathcal{I}^v_{min} F^v_g}{F^v} > 1$, $\mathcal{I}^v_g = \lfloor \frac{\mathcal{I}^v_{min} F^v_g}{F^v} \rfloor$. If $0 < \frac{\mathcal{I}^v_{min} F^v_g}{F^v} < 1$, $\mathcal{I}^v_g = \lceil \frac{\mathcal{I}^v_{min} F^v_g}{F^v} \rceil$. The latter condition is used to instantiate one instance for groups that have a smaller number of flows. This approach works best when the number of flows is large ($\mathcal{I}^v_{min}$ is large).

- *For small numbers of flows:* If the number of flows is small and the objective is to minimize the number of NF instances, the clustering module could be skipped. In this case, all flows will be in one group and $\mathcal{I}^v = \mathcal{I}^v_{min}$.

Second approach: In this approach the minimum number of NF instances needed to serve each group of flows ($g \in \mathcal{G}$) is calculated using Theorem 1. ClusPR groups flows into intra-cluster and inter-cluster sets of flows. If there are k access clusters, there will be $\frac{k(k-1)}{2}$ number of cluster-pairs. Thus, there will be a maximum of k groups of intra-cluster flows and $\frac{k^2-k}{2}$ groups of inter-cluster flows. The maximum number of groups of flows in $\mathcal{G}$ is equal to $\frac{k^2+k}{2}$. $\mathcal{I}^v_{g,min}$ is the minimum number of NF instances of type v needed for serving the set of flows in group g. The maximum total number of NF instances instantiated by ClusPR ($\mathcal{I}^v_{max}$) is a summation of the number of instances instantiated for each of the groups. That is, $\mathcal{I}^v_{max} = \sum_{\forall g} \mathcal{I}^v_{g,min}$.

Theorem 2. *The number of NF instances of type $v \in \mathcal{V}$ instantiated by ClusPR, $\mathcal{I}^v$, is bounded by $\mathcal{I}^v_{min} \leq \mathcal{I}^v \leq \mathcal{I}^v_{min} + \frac{k^2+k}{2}$ where k is the number of access clusters.*

Proof: refer to Appendix B.

From Theorem 2, it can be observed that the number of NF instances created by ClusPR using the second approach is upper bounded. And the upper bound is a function of the number of access clusters.

*2) **Shortest Path**:* The second module in the initialization phase is the shortest path module. The purpose of this module is to obtain path information about the groups of flows by identifying nodes that are on the shortest path of flows. The shortest path between access nodes of the flows is calculated using classical shortest path algorithms such as Dijkstra's algorithm. The nodes that are on the shortest paths are regarded as the "best" candidates for hosting NF instances (as noted in **O1**). In addition, one hop and two hops neighboring nodes of the shortest path nodes are also identified. This is done to increase the number of candidate nodes for hosting NF instances as the shortest path nodes might not have enough resources. The path information captured through the selected shortest path and neighboring nodes is then transferred to the placement phase.

Flows have various service chain requirements. To ensure that NF instances placed on a shortest path node are needed by flows whose shortest path passes through the node, each shortest path node keeps a *list*. The *list* is used to record the different types of services the flows require. In addition, the nodes will have a *weight* that is used to record the number of flows whose shortest path passes through the node. For example, if a node is on the shortest path between three pairs of access nodes that have 3, 5 and 10 flows between them, the node will have a weight equal to 18. In addition, if these flows require DPI, proxy and firewall services, the node will have a list containing these three services.

Shortest path nodes are ordered based on their weight: the higher the weight of a node the higher its priority for hosting NF instances. In other words, nodes that are on the shortest path of many flows are given higher priority to host NF instances, as noted in (**O1**). If the weight of the nodes is equal, then nodes that have higher processing power are given priority over nodes that have lower processing power. Next the NF placement decisions are made for each group of flows.

B. Placement Phase

In this phase, NF instances are placed on the shortest path nodes and/or their neighboring nodes. The type and number of NF instances required to serve each of the groups of flows have been calculated in the initialization phase. The set of *NF types to be placed for a group of flows are ordered according to their popularity, which is measured by the number of flows that require the NF type*. The most popular NFs are prioritized to be placed first, considering (**O3**), with ties broken by prioritizing the NF requiring more processing power. The number of each type of NF to be instantiated is recorded. The placement phase places NF instances for each group of flows.

Algorithm 2 ClusPR's Placement Heuristic

$Q_n \leftarrow$ priority queue of candidate nodes
$Q_{nf} \leftarrow$ priority queue of NF types to be placed
$i_v \leftarrow$ number of instances of NF type v to be placed
$ActiveNode \leftarrow$ `null`
$n.(list)$ list of node $n \in Q_n$
$C \in \{0,1\} \leftarrow C = 1$ if consolidation is used
$T \leftarrow$ per NF utilization threshold for consolidation
while Q_{nf} not empty **do**
    $v \leftarrow$ NF type from top of Q_{nf}
    while Q_n not empty **do**
        if $ActiveNode$ and $v \in ActiveNode.(list)$) **then**
            if C & $ActiveNode$ has v& $\rho^v_n < T$ **then**
                $i_v = i_v - 1$, Continue to next v in Q_{nf}
            else if n has resources **then**
                Place v on $ActiveNode$
                $i_v = i_v - 1$, Continue to next v in Q_{nf}
        else
            $n \leftarrow$ top of Q_n
            if $v \in n.(list)$ **then**
                if C &n has v hosted and $\rho^v_n < T$ **then**
                $i_v = i_v - 1$, continue to next v in Q_{nf}
                else if n has resources **then**
                Place v on n, $ActiveNode \leftarrow n$
                $i_v = i_v - 1$, Continue to next v in Q_{nf}

Bin-Packing: The placement heuristic, summarized in Algorithm 2, does a *bin-packing of the ordered NF types on the set of best candidate shortest path nodes and their one hop and two hops neighboring nodes*. An NF instance is placed on a shortest path node if and only if the node has the NF type in its *list*, which contains a list of the NF types needed by the flows whose shortest paths pass through the node. An NF type that is on top of the priority queue of NF types is taken and the queue of "best" candidate nodes is iterated through until a node that has the NF type in its *list* is found. Once a node is found, it is checked if the node has enough processing and memory capacity to support the NF type.

If all the "best" candidate shortest path nodes do not have enough resources to host the NF type, the algorithm checks for

one hop neighboring nodes of the shortest path nodes followed by two hops neighboring nodes. Once a node is found the NF is placed and the number of instances of the NF type to be placed is decreased by one. The node will then be regarded as an *active node* for placing the next NF type. Nodes that are more than two hops away from the shortest path nodes could also be considered for hosting NFs but the farther the candidate nodes are from the shortest path nodes, the higher the probability that flows will experience larger path stretch by using NFs hosted on these nodes.

Diversity: The placement *heuristic diversifies the types of NFs placed on a node*. That is, the algorithm prioritizes placing different types of NF instances on one node rather than placing multiple instances of the same type of NF on the node. If different types of NFs are placed on one node, the probability that a flow can get all of the services it requires from one node will be high, as noted in (**O2**). Serving a flow's chain in one node has advantages such as decreasing the communication cost and the delay experienced by the flow.

Next, the following NF type is picked from the queue of NFs and placed on the *active node* provided that the NF is found in its *list*. If not the algorithm returns to the queue of the nodes, and looks for another node following the same steps as above. After placing one instance of all types of NFs, the algorithm returns back to the top of the queue of NFs and places the second instances. This process is repeated until all the instances of all NF types are placed.

Consolidation: The aim of the consolidation step is to share placed NF instances among groups of flows to facilitate better utilization of the NF instances. If consolidation is applied, before placing an NF instance at a given node, the algorithm checks if the node has already hosted this type of NF for the other groups of flows. If the estimated utilization of the instance already placed on the node, ρ_n^v, is below a given threshold (e.g.,50%), it is assumed that the instance has enough available capacity to host the flows in the other group as well so a new instance will not be instantiated. Consolidation will result in the instantiation of fewer number of instances.

Higher values of the threshold encourage consolidation thus leading to the instantiation of less number of NF instances. However, this has a risk of increased path stretch and rejection of flows. Comparatively lower threshold values discourage consolidation. If the *first approach* (initialization phase) is used to decide the number of instances, then the instances created are already the minimum number of instances needed to serve the set of flows so *the consolidation step should not be applied*.

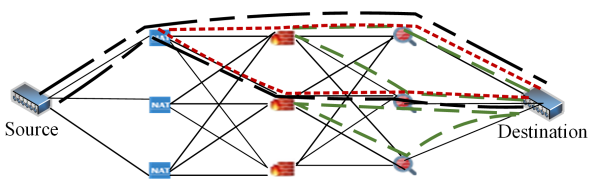


Fig. 2: Example of paths in $\mathcal{R}_{n_j}^k$ and $\mathcal{R}$ for k=2

C. *Routing Phase*

The routing phase is responsible for making flow-level decisions of assigning NF instances to flows and routing flows end-to-end i.e., from their source node through the assigned NF instances of their service chain and finally to their destination node. In making these decisions, two objectives are considered: *satisfying delay requirements of flows* and *load balancing among NF instances*.

A flow's routing problem is *modeled as a multi-stage graph* in which the stages of the multi-stage graph represent the services in the service chain of the flow. The vertexes of a stage of the graph represent the NF instances the flow can be assigned to. At each stage, a flow can be assigned to one of the NF instances that are placed for its group. These are the NF instances placed on the shortest path nodes and their neighboring nodes of the flow's group.

For constructing the multi-stage graph, the costs on the links of the graph also need to be calculated. The costs can be calculated using shortest path algorithms such as Dijkstra's algorithm. The shortest path costs of the links from the source node to the nodes hosting the first NF instance of the chain and the links from the destination node to nodes hosting the last NF of the service chain need to be calculated for each of the flows. The costs of the links between the stages (nodes hosting NFs in the chain) are calculated once, which decreases the computational complexity of constructing the graph. We propose a new algorithm that is based on the ideas of dynamic programming and incorporates novel methods to enable solving the flow routing problem considering both end-to-end delay and the utilization level of instances. Before explaining the algorithm, the dynamic programming based shortest path algorithm is explained for completeness.

To formulate the dynamic program, two distance notations are adopted: $C(n, n^{'})$ and $D(n_j, d_f)$. $C(n, n^{'})$ is used to represent the cost of the shortest distance between nodes n and $n^{'}$ that belong to two consecutive stages (NFs in the chain). e.g., $C(s_f, n_1)$ represents the cost of the shortest distance between the source node of flow f (s_f) and node n_1 that hosts the 1^{st} service instance. $D(n_j, d_f)$ represents the shortest distance between node n_j that is hosting the j^{th} service of the flow to the destination node, e.g., $D(n_2, d_f)$ represents the shortest distance from node n_2 that is hosting the 2^{nd} service to the destination node (d_f).

The dynamic program formulation is given as

$$D(s_f, d_f) = \min_{n_1 \in \mathcal{N}_1} (C(s_f, n_1) + D(n_1, d_f)) \quad (22)$$

$$D(n_j, d_f) = \min_{n_{j+1} \in \mathcal{N}_{j+1}} (C(n_j, n_{(j+1)}) + D(n_{(j+1)}, d_f)) \quad (23)$$

$\mathcal{N}_j$ is the set of nodes that are on the shortest path and one hop and two hops away from the shortest path and are hosting the flow's j^{th} service type for the group the flow belongs to. The dynamic program is solved starting from the destination node until the source node is reached.

*1) **The proposed flow routing algorithm**:* The objectives of the proposed flow routing algorithm are to satisfy the delay requirement of the flow and balance the load among NF instances. To achieve these, the algorithm first *(1) finds a set of routes that satisfy the delay requirement of the flow*, then (2) out of these paths a flow is assigned to a path that has the *minimum maximum NF utilization (to balance the load among NF instances)*.

To find a set of routes that could potentially satisfy the delay requirement of the flow, k shortest paths are saved from a node in a stage of a graph to the destination. That is the distance $D(n_j, d_f)$, which represents the shortest distance between a node n_j that is hosting the j^{th} service of the flow to the destination node, is extended to a set of paths, $\mathcal{R}^k_{n_j}$, which has k elements (k shortest paths from a node n_j of the j^{th} stage to the destination). Fig. 2 shows an example of the paths, which are highlighted, saved for k=2. $\mathcal{R}_{n_j}$ represents a set of paths from a node n_j that is hosting the j^{th} service of flow f to the destination node, so $\mathcal{R}^k_{n_j} \subseteq \mathcal{R}_{n_j}$.

Nodes found in the last stage of the multi-stage graph have a direct link with the destination node, so for each node in the last stage, $n_J \in \mathcal{N}_J$, $\mathcal{R}^k_{n_J}$ will have one element only. The distance from nodes in the last stage, J, to the destination node i.e., $D(n_J, d_f)$) is calculated using shortest path algorithms. A set of paths from nodes in the $J-1$ stage to the destination node ($\mathcal{R}_{n_{J-1}}$) can be calculated as:

$$\mathcal{R}_{n_{J-1}} = \{C(n_{J-1}, n_J) + D(n_J, d_f) : n_J \in \mathcal{N}_J, n_{J-1} \in \mathcal{N}_{J-1}\} \tag{24}$$

The set of k shortest paths from a node in the $J-1$ stage to the destination node ($\mathcal{R}^k_{n_{J-1}}$) is taken from the set $\mathcal{R}_{n_{J-1}}$. For a node in a stage $j \in \{1 \dots J-2\}$, a set of distances ($\mathcal{R}_{n_j}$) can be calculated as

$$\mathcal{R}_{n_j} = \{C(n_j, n_{j+1}) + D(n_{j+1}, d_f) : D(n_{j+1}, d_f) \in \mathcal{R}^k_{n_{j+1}}, n_{j+1} \in \mathcal{N}_{j+1}, n_j \in \mathcal{N}_j\} \tag{25}$$

The set $\mathcal{R}_{n_j}$ is constructed by using the set of k shortest paths to the destination saved in the $j+1^{th}$ stage (i.e., $\mathcal{R}^k_{n_{j+1}}$) and costs between nodes in the stages j and $j+1$ of the multi-stage graph, $C(n_j, n_{j+1})$. Similarly, a set of k shortest distances ($\mathcal{R}^k_{n_j}$) is found from the set $\mathcal{R}_{n_j}$ for each node $n_j \in \mathcal{N}_j$. That is the k shortest paths from all NFs of a stage to the destination node are calculated for the stages one to $J-1$. Finally a set of source to destination end-to-end paths are calculated as:

$$\mathcal{R} = \{C(s_f, n_1) + D(n_1, d_f) : D(n_1, d_f) \in \mathcal{R}^k_{n_1}, n_1 \in \mathcal{N}_1\} \tag{26}$$

If there are N_1 number of nodes in $\mathcal{N}_1$ i.e., the first stage of the graph, with each node having k shortest paths to the destination, there will be in total kN_1 number of source to destination end-to-end paths in the set $\mathcal{R}$.

Out of these paths in $\mathcal{R}$, the set of paths that are able to fulfill the delay requirement of the flow ($\mathcal{R}^d$) are identified. Then, the objective of balancing the load among the NF instances is considered by adopting the min-max fairness. The maximum utilization of the NF instances on each of the routes in $\mathcal{R}^d$ is calculated. The route that has the minimum maximum NF utilization is then chosen for serving and routing the flow. In the situation where there are no routes that can satisfy the delay requirement of the flows ($\mathcal{R}^d$ is empty), the flow is assigned to a route that has the minimum end-to-end delay.

Theorem 3. *ClusPR has a complexity of $O(FN_gL\log N))$, where N_g is the average number of shortest path and neighboring nodes per group. F, N and L are the number of flows, nodes and links respectively.*

Proof: refer to Appendix C.

V. Online Placement and Flow Routing: incremental ClusPR(iClusPR)

ClusPR is an offline algorithm that performs NF placement and flow routing decisions for a set ($\mathcal{F}$) of flows, i.e., the information of all flows is known to ClusPR beforehand. In an online environment, flows will arrive sequentially so resource allocation decisions need to be made for the flows that arrive without knowledge of future incoming flows. iClusPR is an online NFV resource allocation algorithm. It is developed based on ClusPR so it has a design similar to ClusPR (shown in Fig. 1). iClusPR performs dynamic scaling specifically horizontal scaling that is adjusting the number of NF instances depending on the traffic demand. iClusPR makes resource allocation decisions on a time slot basis. That is flows that arrive at a given time will be assigned resources at the subsequent decision time slot. The modifications made in iClusPR for each of the modules are explained below:

A. Initialization phase

1) Clustering: The clustering module of iClusPR serves the same purpose as in ClusPR, that is to group flows based on the proximity of their path. The same clustering algorithm as in ClusPR is used. iClusPR clusters access nodes once when the algorithm is run for the first time on a network topology. Upon arrival of flows, the clustering module simply groups flows based on their source and destination nodes. Then, the NF instances needed to serve each group of flows are calculated using the second approach, which is explained in the initialization phase of ClusPR.

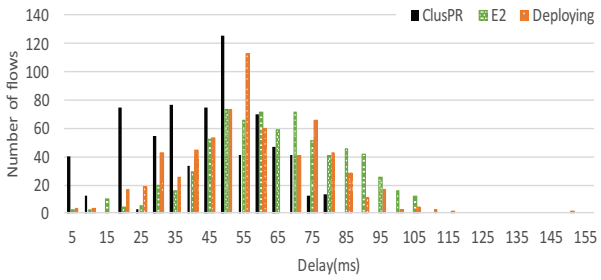
2) Shortest Path: This module of iClusPR is similar to its counterpart in ClusPR and it extracts path information by identifying nodes that are on the shortest path of flows. This calculation needs to be performed upon arrival of flows.

B. Placement phase

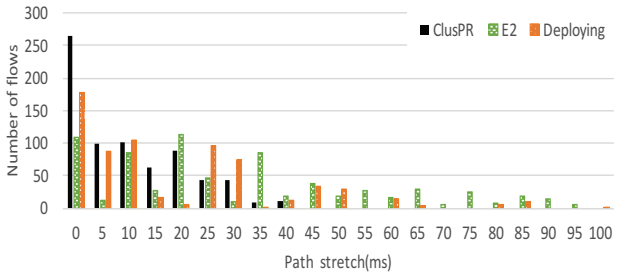
This phase performs *dynamic scaling* by adjusting the number of NF instances instantiated in the network. It takes as input the shortest path nodes and their one hop and two hops neighboring nodes, the NF instances created on these nodes as well as the type and number of NF instances needed to serve each group of flows.

The NF instances instantiated in the previous decision slots can serve the incoming flows as well provided that they have enough available capacity. iClusPR uses a threshold based approach to decide if an NF instance is able to serve the incoming flows or not. That is NF instances whose residual or available capacity is above the threshold value are assumed to have enough available capacity for hosting the incoming flows. Higher threshold values encourage the instantiation of new NF instances and might lead to overprovisioning or underutilization of resources. On the other hand, lower threshold values encourage the use of existing NF instances but could result in over-utilization of resources and rejection of flows.

Similar to ClusPR's placement heuristics, iClusPR orders the set of NF types needed to be created based on their popularity, which is measured by the number of flows that need the NF type. The most popular NF type is prioritized to

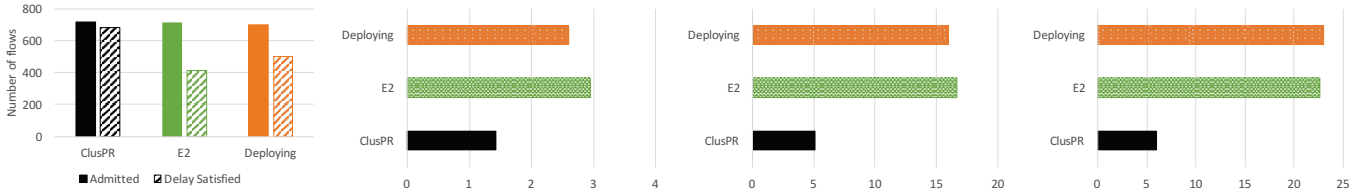


(a) Distribution of total delay



(b) Distribution of path stretch

Fig. 3: Delay performance with service chain length=2



(a) Network utilization, chain = 2 (b) Average delay, chain = 2 (c) Worst delay, chain = 2 (d) Worst delay, chain= 4

Fig. 4: Network utilization, average and worst-case normalized total delays for different chain lengths

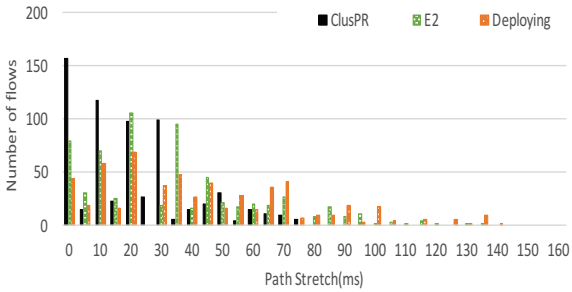


Fig. 5: Path stretch distribution, chain length = 4

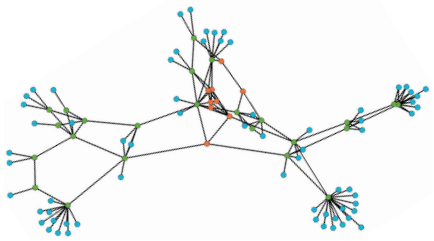


Fig. 6: Rocketfuel topology AS 1221:100 nodes and 294 links

be placed first. The NF type that is on the top of the NF queue is picked first and the *placement heuristic checks whether there is an existing NF of the same type whose residual capacity is above the threshold specified*. If so, the algorithm goes to the next NF type without instantiating a new instance. If there are no existing NF instances of the same type, a node that is on top of the selected nodes queue is picked and a new instance of the NF type is instantiated following the same steps as in ClusPR's placement heuristic. Besides creating new NF instances, an instance might also be removed from the network if all the flows it was serving have departed.

C. Routing phase

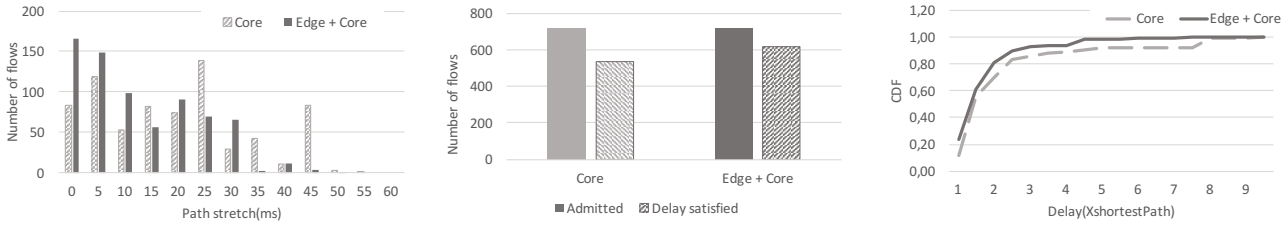
In this phase, flows are assigned NF instances and routed end-to-end that is from their source node through the NF instances of its chain finally to their destination node. The same algorithm as in the routing phase of ClusPR is used.

VI. Experimental Results

We analyze the performance of ClusPR and iClusPR extensively. ClusPR's performance is also compared with two alternatives, E2 [8] and [11] (referred to as "Deploying" here), on realistic networks. We report results on experiments with a practical ISP network topology, the Rocketfuel [23] topology AS 1221 shown in Fig. 6 used as a test network. The nodes in the topology are classified as "access" (in blue), "edge" (green) and "core" (orange) nodes, in a manner similar to [24]. NFs are considered to be hosted on (or adjacent to) edge and core nodes. It is assumed that each host has 4 CPU cores and 8GB memory. Every NF instance requires one CPU core and 2GB memory, with a service rate of 10Mbps. There are five types of NFs (e.g., Firewall, DPI, NAT, IDS, and Proxy).

All the links have a capacity of 1 Gbps, and the delays on the links are: access-edge: 3 ms; edge-core: 10 ms; core-core: 40 ms. The source and destination nodes of flows as well as the services required by the flows are generated randomly. The arrival rate of each flow is assumed to follow a log-normal distribution [25] with an average rate of 0.5 Mbps. The length of the service chain for each flow is assumed to vary in the range of 2 to 4 NFs and the service types in the chain for each flow are selected randomly.

The performance metrics used are total delay, path stretch (measured as the difference between the total delay and shortest path delay), network utilization (measured by the number of flows admitted and also by those whose delay requirement is satisfied), number of NF instances created and the per-NF utilization level. These performance metrics are compared for different setups such as variable chain length and distribution of node processing capacity.



(a) Distribution of path stretch (b) Network utilization (c) CDF of the Normalized Delay

Fig. 7: Effect of using edge computing nodes

A. Evaluation of ClusPR

Total Delay and Path Stretch:

The total delay of a flow is measured as the summation of the delays on the links it is routed through. Fig. 3 shows the total delay and path stretch distributions with 720 flows that have a two NFs long service chain. Both E2 and ClusPR instantiated 74 instances, which is the minimum number of instances needed for the flows.

ClusPR has a shorter path stretch compared to both Deploying [11] and E2 for the *same number of instances*. The performance gain is partly because ClusPR uses the flows' path information in NF placement decision-making and it diversifies the type of NFs placed on a node, thus increasing the probability that a flow can get all of the services of its chain at one node.

Average and Worst-case Delays: Fig. 4b and 4c show the average and worst-case total delays, respectively. They are normalized with respect to the shortest path delay of flows. ClusPR is able to achieve a worst-case normalized delay that is $10\times$ less than the worst-case normalized delay of E2 and Deploying. The average normalized delay of ClusPR is $1.2 - 1.6\times$ less compared to E2 and Deploying.

Network Utilization: To analyze the delay satisfaction of flows, flows are set to have a specified delay requirement in terms of the maximum normalized total delay that they can tolerate. The normalized delay requirement of flows is assumed to be uniformly distributed between $1 - 2.5\times$ their shortest path delay.

Fig. 4a shows the number of flows that are admitted out of the 720 flows and those whose delay requirement is satisfied. ClusPR, E2 and Deploying achieve similar network utilization in terms of the number of admitted flows, but their delay performance differs considerably, which also results in a noticeable difference in terms of the number of flows whose delay is satisfied. The delay difference is due to the following underlying reason. For E2, delay or path stretch is not considered in the heuristic. Deploying, on the other hand, prioritizes maximizing network utilization and does not balance the load across NF instances. ClusPR satisfies the delay requirement of 95% of the flows while E2 and Deploying managed to fulfill the delay requirement of 60% and 70% of the flows, respectively. ClusPR satisfied the delay requirement of $25 - 35\%$ more flows compared to E2 and Deploying.

Effect of Service Chain Length: We now analyze the effect of the service chain length. Fig. 5 demonstrates the path stretch distribution of 650 flows with a service chain length of four. Both E2 and ClusPR instantiated 132 NF instances for serving the flows.

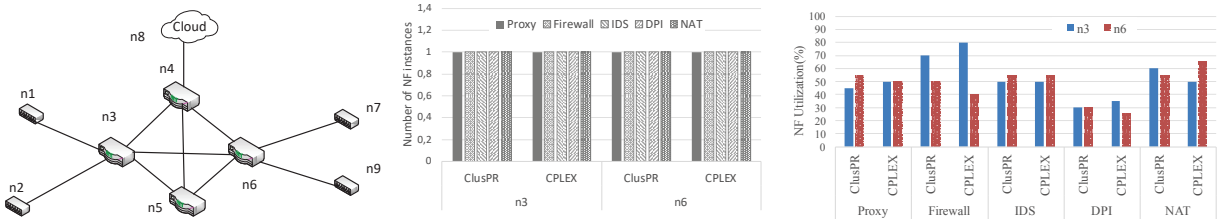
In comparison to the path stretch experienced by flows with service chain length=2 (Fig. 3b), flows with a service chain length of four experienced a larger path stretch. This is because flows with longer service chains need to traverse through multiple NF instances which might not be co-located in the same node. The worst-case delay is shown in Fig. 4d. Compared to the worst-case delay experienced for service chain length of 2 (Fig. 4c), ClusPR's worst-case delay for a chain length of 4 has increased slightly from $5\times$ to $6\times$ the shortest path delay. E2 and Deploying have observed $6\times$ and $7\times$ increase in the worst-case normalized delay. Thus, ClusPR can adapt better to the change in the service chain length compared to both E2 and Deploying.

Effect of edge computing nodes: In this section, the effect of using edge computing nodes as hosts of NF instances is analyzed. Two network setups are considered. In the first setup the processing power is concentrated in three centralized (core) clouds. Each of the clouds has 44 cores. Thus, in total there are 132 CPU cores in the network. In the second setup, some of the processing power of the central clouds is distributed to the edge nodes. The three central cloud nodes have 24 cores each while 30 edge nodes have two cores each. We will refer to this setup as "Edge + Core" and the first setup as "Core". Thus, in total the "Edge + Core" network setup will also have 132 CPU cores.

Fig. 7a shows the path stretch observed by 700 flows in the "Edge + core" and "Core" network setups. The "Edge + core" setup has a better performance as more flows experience zero or a small amount of path stretch compared to the "Core" setup. As can be inferred from the CDF of the normalized delay shown in Fig. 7c, the "Edge + core" setup has a smaller worst-case and average total delays. The number of flows that are admitted and those whose delay requirement is satisfied is shown in Fig. 7b. The delay requirement of 75% and 86% of the flows is satisfied in the "Core" and "Edge + core" setups respectively. These results demonstrate that hosting NF instances on edge computing nodes also has a significant performance gain by decreasing the latency.

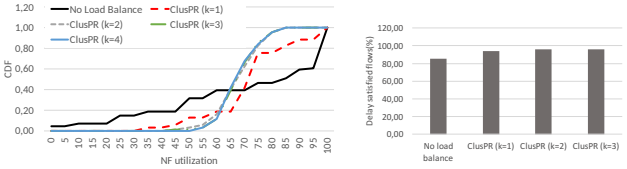
B. Effect of load balancing across NF instances

ClusPR's flow routing algorithm balances the load among the instantiated NF instances. In this section, ClusPR's load balancing approach is analyzed and compared with the no load balancing approach, which assigns to flows NFs that are on



(a) Test network (b) NFs on nodes n3 & n6 (c) Per-NF utilization

Fig. 8: Comparison between ClusPR and CPLEX



(a) CDF of NFs utilization (b) Network utilization

Fig. 9: Effect of load balancing

the shortest path of the multi-stage graph model of the flow, provided the NFs are not 100% utilized. ClusPR's flow routing algorithm saves k shortest paths from a node at a given stage to the destination node, then it balances the load considering the multiple paths saved. k is an algorithmic parameter and the impact of the value of k is also assessed.

In Fig. 9a the utilization of instances when no load balancing is applied is compared with ClusPR with k varying from one to four. When no load balancing is applied, some of the instances have very low utilization while some other instances are highly utilized, going up to 100%. For ClusPR with k=1, there is a better distribution of load among the instances. For higher values of k, the load is balanced across the instances with most instances having a utilization between 65%-75%. There is only a slight difference in the utilization level of instances for k more than 3. As a result of the load balancing, ClusPR is able to satisfy the delay requirement of 8% more flows compared to the no-load balancing approach (Fig. 9b).

C. Comparison of ClusPR and ILP Model

The performance of ClusPR is compared with the proposed ILP model (solved using CPLEX) for a small Test network topology with 9 nodes and 11 links shown in Fig. 8a. Nodes n3, n4, n5, n6 and n8 are able to host NF instances and have a processing capacity of 5,4,4,5 and 10 cores, respectively.

The number and type of instances instantiated for serving 50 flows are shown in Fig. 8b. Two nodes, node n3 and n6, which are on the shortest path of flows are chosen to host NFs by both CPLEX and ClusPR. As can be seen from the figure, ClusPR has created the same number and type of instances as the optimal CPLEX solution. The utilization of the instances is shown in Fig. 8c. ClusPR balances the load across the NF instances and the utilization is only very slightly different from the utilization of the instances in the CPLEX solution. Both ClusPR and CPLEX are able to satisfy the delay requirement of all 50 flows. For the execution time, CPLEX takes 1 hour while ClusPR takes less than 1 second. to get the solution, on the same computer. This indicates that, ClusPR is able to reach a near optimal result, but is much faster.

D. Evaluation of iClusPR

The performance of iClusPR is analyzed and compared with ClusPR, and the effect of the parameter, α, which is a per-NF utilization threshold value, is assessed. In this evaluation, a flow-level simulation was performed.

Simulation setup: The flow arrival process is assumed to be Poisson with an average arrival rate of 1 flow per time unit. The sojourn time of flows is exponentially distributed with an average of 700 time units. The decision time slot is assumed to be 40 time units long. Under these settings, the network can be modeled as an $M/M/\infty$ system. From queuing theory, theoretically, the average number of flows expected in the network is 700.

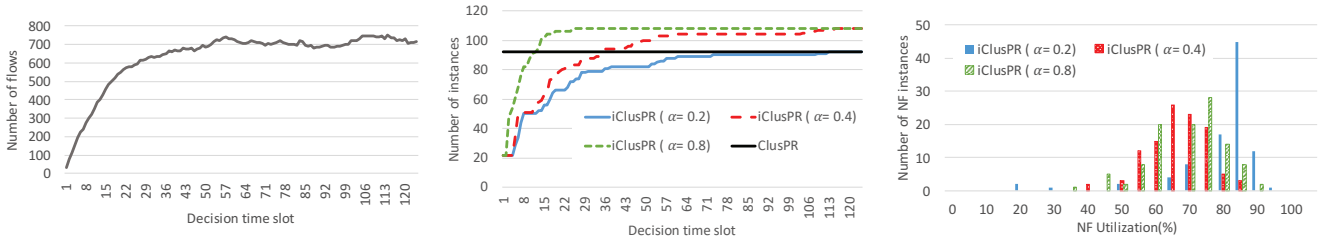
Fig. 10a shows the number of flows in the system at different decision time slots. In total, by time slot 125, 5000 flows have arrived. As expected, the number of flows in the system converges to the theoretical average value i.e., 700. The number of NF instances created for different values of the threshold value (α) is shown in Fig. 10b. For $\alpha = 0.8$, a new NF instance will be instantiated if the existing NF instances have less than 80% available capacity. Thus, the larger the value of α the more the number of instances created. iClusPR also balances the load across the NF instances as can be seen from Fig. 10c. Figs. 11a and 11b show the percentage of flows whose delay requirement is satisfied and the worst-case delays, respectively, for different threshold values. For $\alpha = 0.2$, the network has the lowest performance of all and $\alpha = 0.4$ and 0.8 have almost similar performance. An implication is, α values that are in the middle e.g., $\alpha = 0.4$ or 0.5, give a good trade-off between the number of NF instances and the network performance (delay, utilization).

VII. Related Work and Discussion

In the literature, a number of approaches have been proposed for the NFV-RA problem. A detailed survey can be found in [26]. In the following, we review the most related and recent works.

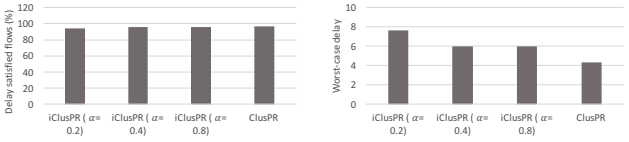
ILP models for the joint NF placement and flow routing problem have been presented in papers such as [27], [28] and [29]. In addition to the models, heuristic algorithms (based on the models) have been proposed in [27] and [28] and a greedy heuristic is proposed in [29]. A shortcoming of the [27] and [29] models is that they do not keep the routing order among services of a chain. In addition, the ILP models are generally not scalable due to the complexity of the problem.

Recently, heurisitc approaches have been proposed to tackle the scalability problem associated with the ILP models. Here



(a) Number of flows (b) Number of instances (c) Per-NF utilization

Fig. 10: Evalutaion of iClusPR for different values of per-NF utilization theshold(α)



(a) Network utilization (b) Worst-case Delay

Fig. 11: Evalutaion of iClusPR

we broadly divide the algorithms into two classes. The first class includes algorithms that avoid path stretch by serving the flows in their path. For example, the resource allocation algorithm in, CoMb [9], [10] and centrality-based heuristics such as [30] and [31] belong in this class. In CoMb, a flow is constrained to use NF instances running in the same node that is found on its path. However, the CoMb approach can considerably limit the utilization of the network since flows are constrained to stay on their path and use a single node for all their services. Relatively, the centrality-based heuristic in [31] has a relaxed restriction as it allows a flow to use NFs placed on more than one node, but still the nodes have to be located on the shortest path of the flow.

In the second class are algorithms that try to increase the utilization of the network disregarding the path stretch. For examples, algorithms proposed in [8], [11], [12], [32] and [33] belong in this class. In [11], referred to as Deploying in this paper, an algorithm that tries to make better use of network resources by promoting flows to reuse instances which have been created instead of instantiating new ones is proposed. Another heuristic approach is the E2 framework [8] which is developed for allocating NF instances and routing flows inside a central office or small data centers. The placement is modeled as a graph partitioning problem and solved using a modified Kernighan-Lin heuristic. Flows are assigned to NFs balancing load across the NF instances.

The proposed schemes, ClusPR and iClusPR, take an approach that can be regarded as being in the middle of these two classes. They do not impose strict restrictions on flows to not deviate from their shortest path. This is because flows might have a relaxed delay requirement which may not be violated even if they deviate from their shortest path. They also do not disregard the effect of path stretch as methods in the second class. ClusPR and iClusPR rather find a balance between minimizing the path stretch, maximizing network utilization and balancing the load among the NF instances. ClusPR and iClusPR are targeted at general networks where resources are distributed in the networks. For networks that have minimal delay between the nodes, as for example in a data-center network, we expect that the performance gain with ClusPR will not be as significant, compared to [8] and [21]. However, ClusPR and iClusPR address the more general problem of having NFs placed at diverse locations, including multiple data-centers across a WAN, which would be required to address scale and diversity typically observed in service provider networks. We believe that ClusPR's ability to consider the trade-off across multiple dimensions will prove invaluable in production networks

VIII. Conclusion

The flexibility brought about by NFV can potentially change the way networks are managed and services are provisioned. However, efficient resource allocation algorithms are needed to instantiate NF instances when and where needed, and route flows through them accordingly. A comprehensive ILP model is provided for the NFV-RA problem. Based on the useful insights obtained from the optimal solution of the ILP model, we develop an offline heuristic algorithm, ClusPR, that is scalable and balances across multiple objectives. In addition, an online algorithm, iClusPR, that dynamically adjusts the number of NFs depending on the traffic demand and network state is presented. By factoring in information about the path of flows into the NF placement decision making and diversifying the type of NFs placed on a node, ClusPR and iClusPR are able to considerably minimize the path stretch and maximize the network utilization while balancing the load across NF instances. Compared to the state-of-the-art approaches, ClusPR manages to decrease the average normalized delay by a factor of $1.2-1.6\times$ and the worst-case delay by $10\times$, while admitting the same or slightly larger number of flows. At the same time, ClusPR satisfies the delay requirement of 25-35% more flows and balances the load across NF instances. The online algorithm iClusPR is also able to perform dynamic NF scaling while having performance that is comparable to that of ClusPR.

Appendix A

Proof of Theorem 1:

In order to have a stable system, a server (NF instance) should not be loaded more than its service rate. The aggregate arrival rate of flows that require an NF type v is equal to $F^v\lambda$. The total service rate of v type NF instances should be more than the aggregate arrival rate of the flows, that is $F^v\lambda < \mathcal{I}^v_{min}\mu^v$. Where $\mathcal{I}^v_{min}$ is the minimum number of v type NF instances. Thus, $\mathcal{I}^v_{min} = \lceil \frac{F^v\lambda}{\mu^v} \rceil$

■

APPENDIX B

Proof of Theorem 2:

For each of the group of flows $g \in \mathcal{G}$, ClusPR calculates the minimum number of NF instances needed to serve the flows using Theorem 1. The maximum total number of NF instances instantiated is a summation of the minimum number of NF instances calculated for each of the groups, that is $\mathcal{I}^v_{max} = \sum_{\forall g} \lceil \frac{F^v_g \lambda}{\mu^v} \rceil$, F^v_g is the number of flows in group g that require NF type v. The maximum value $\mathcal{I}^v_{max}$ can take is $\sum_{\forall g} (\frac{F^v_g \lambda}{\mu^v} + 1)$. Where $\frac{F^v_g \lambda}{\mu^v}$ is an integer quotient of the float division. Since there are a maximum of $\frac{k^2+k}{2}$ number of groups.

$$\mathcal{I}^v_{max} = \sum_{\forall g} (\frac{F^v_g \lambda}{\mu^v} + 1) = \frac{\sum_{\forall g} F^v_g \lambda}{\mu^v} + \frac{k^2+k}{2}. \tag{27}$$

since $\sum_{\forall g} F^v_g = F^v$,

$$\mathcal{I}^v_{max} = \frac{F^v \lambda}{\mu^v} + \frac{k^2+k}{2} = \mathcal{I}^v_{min} + \frac{k^2+k}{2} \tag{28}$$

According to Theorem 1, the minimum number of instances that need to be instantiated to serve the set of flows ($\mathcal{F}$) is $\mathcal{I}^v_{min}$. Thus, the number of NF instances instantiated by ClusPR($\mathcal{I}^v$) is bounded by

$$\mathcal{I}^v_{min} \leq \mathcal{I}^v \leq \mathcal{I}^v_{min} + \frac{k^2+k}{2} \tag{29}$$

■

APPENDIX C

Proof of Theorem 3:

The initialization phase has two modules these are clustering and shortest path. The clustering module uses Kruskal's algorithm which has a complexity of O($L \log L$). The shortest path utilizes Dijkstra's shortest path algorithm which has a complexity of O($L \log N$). Thus the complexity of the initialization phase is *O($L \log L$ + $L \log N$)*.

The complexity of the placement heuristics depends on, the number of flow groups, number of instances to be placed and the number of nodes that can host NFs. From Theorem 1, the number of instances to be placed can roughly be approximated by the number of flows (F). The placement heuristic is run for each of the group of flows. For each group, ClusPR utilizes the nodes that are on the shortest path of flows and their neighboring nodes as candidate nodes for placing NF instances. Let N_g be the number of these candidate nodes and G be the total number of groups. Thus, the complexity of the placement heuristics will be *O(GFN_g)*.

The routing of a flow is modeled as a multistage graph. The maximum number of vertices of the graph is equal to JN_g where J the number of stages of the graph and N_g is the number of candidate nodes. The number of edges between the stages of the graph is equal to $N_g(N_g-1)(J-1)$. In addition, there will be $2N_g$ edges between the source node of the flow and the nodes in the first stage and the destination node and nodes in the last stage of the graph. Simplifying, in total the multi-stage graph will have $N_g^2+N_g$ edges. Dijkstra's shortest path algorithm is used to find the cost of the edges, which will have a complexity O($L \log N(FN_g + N_g^2)$).

ClusPR's routing algorithm has complexity proportional to the complexity of a dynamic programming shortest path algorithm whose complexity is in the order of the summation of the number of edges and vertices of the multi-stage graph. Since the graph is constructed for each of the flows, the complexity of solving the multi-stage graph for all of the flows will be O($F(N_g^2+N_g+JN_g)$). Thus, the complexity of the routing algorithms is *O($L \log N(FN_g+N_g^2)+F(N_g^2+N_g+JN_g)$)*. The complexity of ClusPR will be a summation of the complexity of the three phases and can be simplified to *O($FN_gL \log N$)*. ■

ACKNOWLEDGMENT

The work for this paper was performed in the context of the EU FP7 Marie Curie Actions project Grant Agreement No. 607584 (the CleanSky project).

REFERENCES

[1] Y. Woldeyohannes *et al.*, "A scalable resource allocation scheme for NFV: Balancing utilization and path stretch," in *Innovations in Clouds, Internet and Networks (ICIN), 2018 21th Conference on*. IEEE, 2018.

[2] J. Sherry *et al.*, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[3] W. Zhang *et al.*, "Opennetvm: A platform for high performance network service chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*. ACM, 2016, pp. 26–31.

[4] Y. Zhang *et al.*, "Steering: A software-defined networking for inline service chaining," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.

[5] A. L. Andreas Lemke, "Why service providers need an NFV platform: Strategic white paper," Tech. Rep., January, 2015.

[6] A. Gember *et al.*, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," Technical Report, Tech. Rep., 2013.

[7] M. Xia *et al.*, "Network function placement for NFV chaining in packet/optical data centers," in *Optical Communication (ECOC), 2014 European Conference on*. IEEE, 2014, pp. 1–3.

[8] S. Palkar *et al.*, "E2: a framework for NFV applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015.

[9] V. Sekar *et al.*, "Design and implementation of a consolidated middlebox architecture," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012.

[10] Y. Sang *et al.*, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Proceedings of IEEE INFOCOM 2017*. IEEE, 2017, pp. 829–837.

[11] T.-W. Kuo *et al.*, "Deploying chains of virtual network functions: On the relation between link and server usage," in *INFOCOM, 2016 Proceedings IEEE*. IEEE, 2016.

[12] S. Khebbache *et al.*, "Virtualized network functions chaining and routing algorithms," *Computer Networks*, vol. 114, pp. 95–110, 2017.

[13] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 7–13.

[14] Z. A. Qazi *et al.*, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[15] W. Zhang *et al.*, "SDNFV: flexible and dynamic software defined control of an application-and flow-aware data plane," in *Proceedings of the 17th International Middleware Conference*. ACM, 2016, p. 2.

[16] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, vol. 41, no. 6, pp. 853–862, 2010.

[17] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[18] C. T. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Transactions on computers*, vol. 100, no. 1, pp. 68–86, 1971.

[19] O. Grygorash *et al.*, "Minimum spanning tree based clustering algorithms," in *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*. IEEE, 2006, pp. 73–81.

[20] N. Bolshakova and F. Azuaje, "Cluster validation techniques for genome expression data," *Signal processing*, vol. 83, no. 4, pp. 825–833, 2003.

[21] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of cybernetics*, vol. 3, no. 3, pp. 32–57, 1974.

[22] U. Maulik *et al.*, "Performance evaluation of some clustering algorithms and validity indices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1650–1654, 2002.

[23] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[24] A. Afanasyev *et al.*, "Interest flooding attack and countermeasures in named data networking," in *IFIP Networking Conference, 2013*. IEEE, 2013, pp. 1–9.

[25] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 309–322.

[26] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[27] B. Addis *et al.*, "Virtual network functions placement and routing optimization," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 171–177.

[28] A. Mohammadkhan *et al.*, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*. IEEE, 2015, pp. 1–6.

[29] M. C. Luizelli *et al.*, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 98–106.

[30] M. Bouet *et al.*, "Cost-based placement of vdpi functions in NFV infrastructures," *International Journal of Network Management*, vol. 25, no. 6, pp. 490–506, 2015.

[31] S. Ahvar *et al.*, "CCVP: Cost-efficient centrality-based VNF placement and chaining algorithm for network service provisioning," in *Network Softwarization (NetSoft), 2017 IEEE Conference on*.

[32] M. C. Luizelli *et al.*, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, 2016.

[33] M. Mechtri *et al.*, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, 2016.

Yordanos Tibebu Woldeyohannes Yordanos Tibebu Woldeyohannes received the B.S. degree in Electrical Engineering from Mekelle University, Ethiopia in 2007 and M.S degree in Telecommunication Engineering (cum laude) from University of Trento, Italy in 2013. From 2008 to 2010 she was with ZTE corporation Ethiopian branch. Yordanos is currently pursuing the PhD degree in Telematics with the department of Information Security and Communication Technology, NTNU, Norway. Her current research work and interest focuses on mathematical modeling and algorithm design (optimization), resource allocation, network function virtualization and software defined networks. She is also interested in AI, machine learning and wireless networks.

Ali Mohammadkhan was born in Tehran, Iran in 1987. He received the B.Sc. degree in computer engineering from the University of Tehran, Iran, in 2010. In 2013, he obtained his M.Sc. degree from Sharif University of Technology, Tehran, Iran. Ali is currently a fifth-year PhD. candidate in computer science department at the University of California, Riverside. His current research interests are in the areas of computer networks and distributed systems with a special focus on software-defined networking, network function virtualization, and cellular networks.

Dr. K. K. Ramakrishnan is Professor of Computer Science and Engineering at the University of California, Riverside. Previously, he was a Distinguished Member of Technical Staff at AT& T Labs-Research. He joined AT& T Bell Labs in 1994 and was with AT& T Labs-Research since its inception in 1996. Prior to 1994, he was a Technical Director and Consulting Engineer in Networking at Digital Equipment Corporation. Between 2000 and 2002, he was at TeraOptic Networks, Inc., as Founder and Vice President.

Dr. Ramakrishnan is an ACM Fellow, an IEEE Fellow and an AT& T Fellow, recognized for his fundamental contributions on communication networks, including his work on congestion control, traffic management and VPN services. His work on the "DECbit" congestion avoidance protocol received the ACM Sigcomm Test of Time Paper Award in 2006. He has published nearly 250 papers and has 170 patents issued in his name. K.K. has been on the editorial board of several journals and has served as the TPC Chair and General Chair for several networking conferences. K. K. received his MTech from the Indian Institute of Science (1978), MS (1981) and Ph.D. (1983) in Computer Science from the University of Maryland, College Park, USA.

Yuming Jiang has been a Professor with NTNU, Norwegian University of Science and Technology, Trondheim, Norway, since 2005. He received his BSc degree from Peking University and PhD degree from the National University of Singapore. From 1996 to 1997, he worked with Motorola, Beijing, China, and from 2001 to 2003, with the Institute for Infocomm Research (I2R), Singapore. He visited Northwestern University from 2009 to 2010, and Columbia University from 2015 to 2016. He is recipient of a fellowship from the European Research Consortium for Informatics and Mathematics (ERCIM). He was Co-Chair of IEEE Globecom 2005 - General Conference Symposium, TPC Co-Chair of 67th IEEE Vehicular Technology Conference (VTC) 2008, General/TPC Co-Chair of International Symposium on Wireless Communication Systems (ISWCS) 2007-2010, General Chair of IFIP Networking 2014 Conference, and Chair of the inaugurating 2018 International Workshop on Network Calculus and Applications (NetCal 2018). He is first author of the book "Stochastic Network Calculus". His research interests are the provision, analysis, and management of quality of service guarantees, with a particular focus on (stochastic) network calculus and its applications.