Routing via Functions in Virtual Networks: The Curse of Choices

Thi-Minh Nguyen, André Girard, Catherine Rosenberg, *Fellow, IEEE*, and Serge Fdida, *Senior Member, IEEE, Member, ACM*

***Abstract*—An important evolution of the users' needs is represented by the on-demand access to the network, storage, and compute resources in order to dynamically match the level of resource consumption with their service requirements. The response of the network providers is to transition to an architecture based on softwarization and cloudification of the network functions. This is the rationale for the deployment of network functions virtualization (NFV) where virtual network functions (VNFs) may be chained together to create network services. Efficient online routing of demand across nodes handling the functions involved in a given service chain is the novel problem that we address in this paper. We provide an original formulation of this problem that includes link and CPU capacity constraints and is based on the construction of an expanded network. We derive the exact mathematical formulation and propose several heuristic algorithms taking into account the main system's parameters. We conclude by deriving some interesting insights both about the algorithms and the network performance by comparing the heuristics with the exact solutions.**

***Index Terms*—Computer networks, network function virtualization, communication system traffic.**

I. Introduction

THE software implementation of network functions, based on virtualization techniques, will help network operators meet the evolving customer requirements while controlling capital and operational costs. The flexibility of this approach makes it possible to closely match the resource usage with these requirements. As a consequence, network functions that were usually embedded in hardware devices are now virtualized and hosted in the nodes of the network virtual infrastructure. A network service can therefore be decomposed into a Service Chain, namely an ordered sequence of Virtual Network Functions (VNF), which can run on several standard physical nodes. These functions will exist in more than one node and could migrate from one node to another over time, depending on operational criteria. The demand from a user will result in a flow from a source to a destination across nodes hosting the functions involved in the Service Chain, respecting their precedence constraints. This raises important new issues related to efficient online routing in NFV.

Manuscript received June 20, 2017; revised June 8, 2018 and March 7, 2019; accepted April 13, 2019; approved by IEEE/ACM Transactions on Networking Editor S. Chong. Date of publication May 3, 2019; date of current version June 14, 2019. *(Corresponding author: André Girard.)*

T.-M. Nguyen and S. Fdida are with the LIP6 Laboratory, Sorbonne Université, 75006 Paris, France.

A. Girard is with INRS-EMT, Montréal, QC H5A 1K6, Canada, and also with GERAD, Université de Montréal Montreal, QC H3T 1J4, Canada (e-mail: andre.girard@gerad.ca).

C. Rosenberg is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada.

Digital Object Identifier 10.1109/TNET.2019.2912717

II. Related Work

The emergence of Network Function Virtualization (NFV) in 2012 [1] attracted a large amount of work in order to capture the right level of abstraction and to design efficient algorithms related to this approach. By decoupling the functions, called Virtual Network Function (VNF), i.e., firewall, deep packet inspection, load balancing etc., from dedicated hardware appliances and middleboxes, a NFV-enabled network promises an overall cost reduction. Among various issues, the placement of VNFs has been the subject of much work.

A first approach is to consider a static problem from the point of view of connections.[1] Most papers on this topic assume that there is a given set of requests and that the system resources need to be orchestrated to meet these requests. The surveys [2], [3] provide valuable references to relevant previous work. Some mathematical models, including linear and non-linear programming, have been proposed in various papers [4]–[14]. They also provide a vast number of solution heuristics. These models are based on a snapshot of the network at a given instant and are not usually evaluated in a dynamic setting.

A different approach deals with the *online* version. Some of these techniques operate at the packet level. One example is that of [15] which proposes a scheduling technique for packet processing. Traffic flows can be split on different paths and require measurements of the instantaneous loads on equipment. A similar proposal can be found in [16] where new copies of computing functions can be created to respond to load changes and flows can be split to take advantage of these new functions. These models are best-effort and do not guarantee a certain level of performance to the connections.

Other models deal with the admission and routing of the requests themselves where requests arrive at anytime to the system. Whenever a request arrives, user traffic must be steered along a path that starts at the source and visits the nodes where the required virtual functions are implemented, in the order in which they must be applied, before reaching the destination. The online version is studied in [17] in which a new standby/accept service model is introduced. In [18], this problem is studied as the "node-constrained service chain routing problem", and a layered graph model is used to develop

[1]We use the terms connection and demand interchangeably.

conventional routing algorithms. However, these models do not consider capacity constraints.

Another variant of this routing problem was considered in [19], and an algorithm that balances the length of the service function path and the load of service function instances was presented without again taking capacity constraints into account. Similarly, Bhat and Rouskas [20] provided a suite of algorithms that use a variety of solution approaches for tackling the shortest path tour problem by applying the layered graph model in [18], and then developed a new algorithm.

Most recently, Even *et al.* [21] provided a randomized approximation algorithm for path computation and function placement. They followed the framework in [17] for the specification of the processing requirements and routing of requests via processing-and-routing graphs (PR-graphs). They proposed a randomized rounding procedure but no experiment is reported.

III. Our Work

In this paper, we consider a network where connection requests arrive randomly with random holding times. For each connection request with given origin and destination, we want to find a path through the network that will meet a certain number of requirements. If this is not possible, the connection is blocked.

The first requirement is that each connection must use a specific *Service Chain*, i.e., a given set of functions, in a prescribed order, yielding *precedence* constraints. Multiple copies of these functions are placed a priori in some subset of nodes, called the *function* nodes. This could yield loops, i.e., a path might have to visit the same node multiple times and handling loops is one of the challenges we face.

A second set of essential requirements is that the connection will be allocated enough resources along the path to receive an adequate service. First, a connection needs a certain amount of link bandwidth, expressed say in Mbits/sec, for the transmission of packets between nodes. Also, whenever it goes through a node, it also needs a certain processing load, expressed for instance in Mflops/sec. This load has two components: A small one, which is the amount of work that the cpu has to do simply to process the packets that go through the node and a larger one, which is the work the cpu has to do to compute some function if we have decided that this function is to be evaluated in this node.

A third set of requirements, which is a natural by-product of the above, is that the capacity constraints of the different network elements, i.e., the links and the nodes, need to be met at all times. Working on a capacitated network makes the optimization problem very hard to solve quickly.

Working at the connection level has some advantages. First, it is a form of preventive congestion control in the sense that we will not accept the connection if it is expected to cause problems. This is in contrast to some of the packet-level methods described in Section II that will operate once congestion is detected.

Once the connection has been put in place, there is no need to monitor it since the bandwidth and processing requirements have been computed in such a way that the connection will provide a good quality at the packet level. Also, the information needed to make these decisions is quite small and can be updated in real time with little communication overhead, as discussed in Section IV.

One might argue that the network may not be well utilized since the flow cannot be split or re-assigned since it is constrained to follow a single path. This means that there could be some unused capacity available that is left idle. While this is true when we consider a given connection, this potentially unused capacity may very well be used by a new connection request. In other word, the approach we take here uses the *connection* dynamics to spread the load and does not try to do this by splitting flows or re-assigning packets. Which of these approaches is better is an interesting topic that is discussed in Section IX.

Our focus is to design an algorithm that will find a path that meets all the constraints very quickly for each request in a large network. The time limit is due to the fact that the requests arrival rate may be large and a decision has to be taken fast enough that there is no build-up of requests waiting to be connected.

In this paper, we consider networks where the operator can choose to reject connection requests if they require more resources than the network can provide. With this form of admission control, the main performance measure is the connection blocking probability averaged over a large set of requests. The goal is thus to find paths that will produce as low a blocking probability as possible. This we call the *network performance* problem.

There are several options to formulate and solve this problem. One possibility would be to formulate the problem that minimizes blocking subject to resource constraints. This has been done for circuit-switched networks by decomposing the global network problem into separate link problems, optimizing the link blocking separately and combining the results to yield an approximation for the network blocking. This works well for very simple classes of routing, such as random sharing of the load, or hierarchical systems, where it is possible to compute the link arrival processes based on the network traffic flows and capacities. No such work exists for state-dependent routing, such as the one considered here, and finding and evaluating link approximations is a difficult fundamental issue that will require much work.

Another option would be to formulate the problem as a Markov decision process under suitable assumptions. This is intractable, even for simple routing techniques, because of the size of the state space. One possible approach would be to find approximations by state aggregation or decomposition into sub-problems. This approach has never been attempted for simple circuit-switching and it will most likely require a significant theoretical advance in order to be practical for state-dependent routing.

Finally, another option could be to try to find *bounds* on the blocking. Here, the standard decomposition techniques of nonlinear optimization may prove useful if the stochastic routing problem can be cast in that form.

A more practical approach is to design some heuristic routing algorithm that will operate each time a request arrives.

TABLE I
NOTATION

$\mathcal{N}$ (N)	set of nodes (# of nodes)
$\mathcal{L}$ (L)	set of directed links (# of links)
$C_{i,j}$	(integer) residual transmission capacity of the directed link (i, j)
P_i	(integer) residual processing capacity of node i
c	a connection
$\mathcal{F}_c$ (K_c)	sequence of functions $f_1, \dots f_{K_c}$ required by c (# of functions)
$\mathcal{A}_{k,c}$ $(n_{k,c})$	set of nodes containing f_k when c arrives (corresponding # of nodes)
b_c	bandwidth requirement of c
p_c	processing requirement of c in each traversed node
$q_{k,c}$	processing requirement of function f_k in a node for c

This we call the *routing* problem. If we consider a single request, we call this a snapshot problem and we could argue that any path that meets the requirements will do, so that a heuristic could be limited to finding a feasible path. Note however that our real objective is to keep the network blocking probability low. For this reason, most heuristics try to find a path that optimizes a surrogate objective, since the real objective of the network performance problem cannot be computed.

In this paper, we examine a number of such routing heuristics. The quality of each heuristic is evaluated first by the blocking probability that it produces in the network in a dynamic setting. In addition, we examine the average path length it produces and the average computation time per request.

This paper offers a number of significant contributions for the solution of the routing problem.

1) We propose a framework that operates on connections, takes into account CPU and link capacity constraints, implements admission control and is based on the use of expanded networks introduced in Section V to take into account the precedence constraints. Blocking probability is our main quality measure.
2) In Section VI, we give a precise mathematical formulation of the routing problem as an integer minimum-cost flow model with side constraints on the expanded network. We then provide in that section the Lagrangian decomposition of the problem.
3) Because it is unlikely that we can compute exact solutions to the routing problem in the required time, we propose, in Section VII, four fast heuristic algorithms based on this Lagrangian relaxation. The Lagrangian framework allows us to compare algorithms that differ both by the cost functions that are used and by the number of dual iterations that they use.
4) We evaluate in Section VIII the blocking probability of these heuristic algorithms in a dynamic setting and compare them to a straightforward heuristic called GreedyForward.
5) We use the exact model to estimate in Section VIII-I the accuracy of the heuristics.

The main results/messages that we have found are:

- A connection-level framework, along with admission control, is necessary to guarantee quality of service to connections. This implies that capacity constraints are explicitly taken into account which makes the routing problem hard to solve.
- The choice of surrogate objective used for the routing problem makes a significant difference in the blocking probability. In particular, min-hop routing is generally not the best choice.
- A solution based on a single iteration of the Lagrangian relaxation performs significantly better than the heuristics based on min-hop.
- At high blocking, the optimal solution can do significantly better than the heuristics.
- The computation time of the heuristics is at least ten times lower than the one needed to compute the optimal solution.

IV. The System Model

We consider a network, called the *real* or *physical* network, defined by its set of nodes $\mathcal{N}$ with $|\mathcal{N}| = N$, and its set of links $\mathcal{L}$. There is also a given set of functions $\mathcal{F} = \{f_1, \dots, f_F\}$ and a number of copies of each function f_k are placed a priori in some subset $\mathcal{A}_k$ of nodes, called *function* nodes, with $|\mathcal{A}_k| = n_k$. A function f_k can only be computed once in $\mathcal{A}_k$.

Connection requests arrive randomly following a Poisson process with rate λ and an exponential random holding time of mean τ. This means that there cannot be simultaneous arrivals by the definition of a Poisson process. Each connection c carries traffic and has a number of requirements, either on the links or the nodes. First, it needs a certain amount of link bandwidth b_c on each link it uses. This could be a fixed value, for instance for CBR sources, or the effective bandwidth [22] for variable rate sources. Also, whenever connection c goes through a node, it also needs a processing capacity p_c to process the packets through the node. Finally, connection c must use a given sequence of functions $\mathcal{F}_c = \{f_1(c), \dots, f_{K_c}(c)\}$, in that prescribed order, called the *Service Chain*, yielding *precedence* constraints. This entails an additional processing capacity $q_k(c)$ whenever a function $f_k \in \mathcal{F}_c$ is computed in some node. The notation is summarized in Table I. For each connection c with given origin and destination $o(c)$ and $d(c)$, we want to find a path through the network that will meet all requirements and visit the function nodes in the prescribed order.

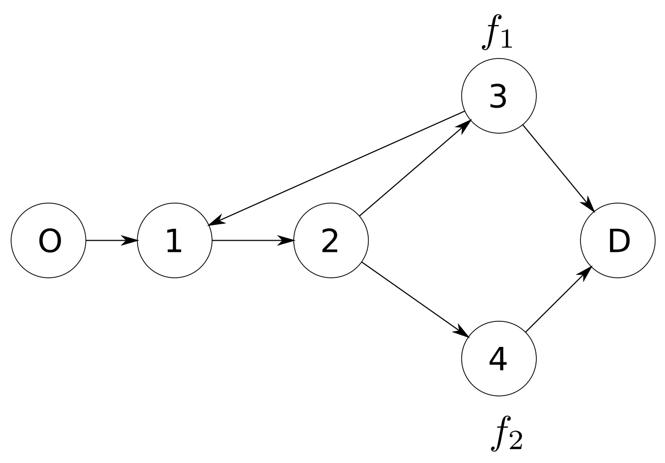


Fig. 1. Capacity constraints and the possible need for loops.

We assume that when a connection request c arrives, we know the residual capacity of all the links and nodes denoted $C_{i,j}(c)$ and $P_i(c)$. From this, we can exclude from the network the links where the available residual transmission capacity is smaller than b_c or the residual processing capacity is smaller than p_c, since they cannot carry the connection. We call this *pruning* the network.

The changes in capacity when a connection arrives or ends can be transmitted to the controller by each node involved in the connection so that it can be upgraded incrementally.

Let $\mathcal{R}(c)$ be the pruned network at the time the connection request for c arrives where $\mathcal{N}_{\mathcal{R}}(c)$ is the set of nodes and $\mathcal{L}_{\mathcal{R}}(c)$, the set of links in $\mathcal{R}$. We describe next the notion of expanded network which will be used to find a path for a new connection request c.

V. The Expanded Network

In this paper, we will be using a solution technique similar to that of [17], [20] based on an *expanded* network built out from the real network.

A. Capacity Constraints and Loops

One might think that pruning the network would guarantee that a connection can be routed. In fact, this is not the case, as we can see from Figure 1, where the functions to visit are f_1 followed by f_2, and this for two reasons.

The first reason is that the precedence constraints on the execution of the functions may be possible only for a path with loops. This is the case in the example, where we need the path $(O, 1)$, $(1, 2)$, $(2, 3)$, $(3, 1)$, $(1, 2)$, $(2, 4)$, $(4, D)$ where the connection needs to go twice through link $(1, 2)$.

Remark 1: This example shows that we *cannot rule out loops* in this kind of problems. This can happen simply because we need to execute all the functions or, in other cases, because they have to be computed in a prescribed order.
As a consequence, we cannot use standard shortest-path algorithms and shortest path algorithms with loops are much more complicated [20].

The second reason is that we cannot assume that the connection can be routed even if the network has been pruned.

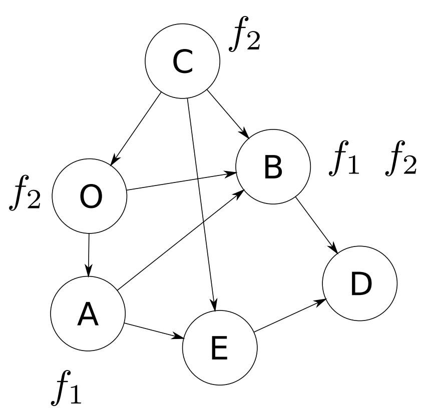


Fig. 2. The pruned network when connection request c arrives.

Assume that when the connection arrives, the available bandwidth is 1.5 units on all links and the connection needs 1 unit per used link. After the pruning, all the links would be seen as feasible. Because we need to use link $(2, 1)$ twice, we need 2 units on link $(1, 2)$ which is not possible. In other words, we need to define a model with explicit capacity constraints. Note that we do not know ahead of time how many times a path would have to go via a link or a node so that we cannot prune more than we do. Thus we cannot avoid *explicit capacity constraints* on the links and the nodes.

B. Building the Expanded Network

Instead of looking for paths with loops in the real network, we build a larger network, called the *expanded* network, in such a way that we can route connections on loop-less paths on this network so that we can use standard path finding algorithms.

The main difficulty with the problem of finding a path for connection c in $\mathcal{R}(c)$ is how to model the precedence constraints. The technique we have used is based on the following argument similar to the technique of [21]. Consider a connection c that arrives to the network at some node $o(c)$. One could route it through a number of intermediate nodes before arriving at one of the nodes that contain $f_1(c)$. The connection can then be routed through a second set of intermediate nodes before arriving at one of the nodes that contain $f_2(c)$. The procedure is repeated until the connection finally reaches $d(c)$.

This suggests that the overall path from $o(c)$ to $d(c)$ can be viewed as a set of sub-paths, or segments, whose collection makes up the complete path. These segments are connected by decision points where we choose which nodes will compute the functions in $\mathcal{F}_c$.

The expanded network $\mathcal{R}_e(c)$ built out of $\mathcal{R}(c)$ is defined by its set of nodes $\mathcal{N}_e(c)$ and its set of links $\mathcal{L}_e(c)$. Note that we need to build a new expanded network for each new connection request.

As an example, consider the pruned network of Figure 2 where we have two functions: f_1, which can be computed in nodes A and B, and f_2, in nodes O, B and C. A connection request c arrives at node O with destination D.

The first segment of the path has to go from node O to either node A or B using any set of links in the network. This first

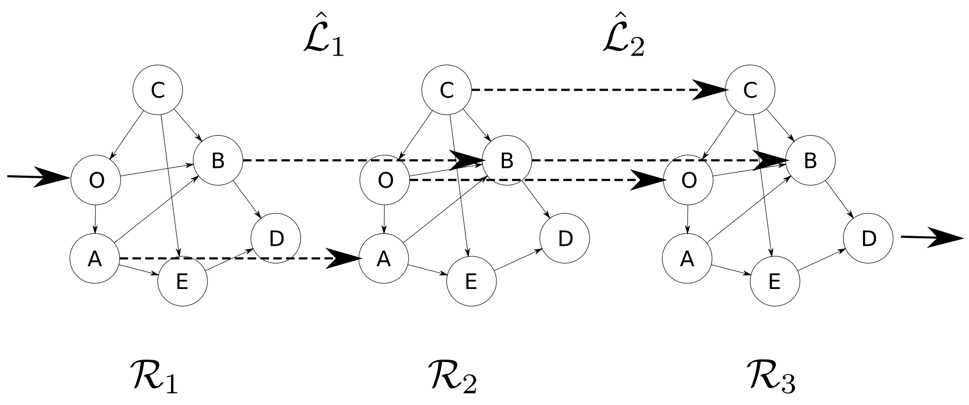


Fig. 3. Expanded network for c from O to D where the index c is omitted for simplicity.

part is modelled by the first sub-network $\mathcal{R}_1$ in Figure 3. Once the connection has reached either node A or B, it has to go through the second segment to get to one of the nodes that can compute f_2, either O, B or C. This second segment is modelled in Figure 3 by the second sub-network $\mathcal{R}_2$. These first two sub-networks are connected by the artificial links represented by the dotted lines. These links show that if the connection reaches node A in segment $\mathcal{R}_1$, then it must enter segment $\mathcal{R}_2$ at node A, and similarly for B. The connection will then be routed in segment $\mathcal{R}_2$ from node A or B, as the case may be, to either node B, i.e., both functions are computed in the same node, or to nodes O or C to compute f_2 there. Once the second function has been computed by O, B or C, the connection has to be routed to D in the last segment $\mathcal{R}_3$.

We can see from this figure that a given node or link of the real network can appear in many places in the expanded network. Also, by construction, any path from node O in $\mathcal{R}_1$ to node D in $\mathcal{R}_3$ will pick up each function only once and in the right order since the only way to go from $\mathcal{R}_k$ to $\mathcal{R}_{k+1}$ is through one of the artificial links.

The general construction procedure can be summarized as follows. First, we construct K_c+1 *copies* of the real network. Copy $\mathcal{R}_k(c)$ is simply the set of nodes and links in copy k of the original network $\mathcal{R}(c)$. These are denoted $\mathcal{N}_k(c)$ and $\mathcal{L}_k(c)$ when we want to be specific as to which copy we are referring to.

For each $\mathcal{R}_k(c)$, we also define a set $\hat{\mathcal{L}}_k(c)$ of artificial links to connect $\mathcal{R}_k(c)$ and $\mathcal{R}_{k+1}(c)$. An artificial link can be defined for each node $i \in \mathcal{R}_k(c)$ that carries function f_k. This link connects $i \in \mathcal{R}_k(c)$ to $i \in \mathcal{R}_{k+1}(c)$. The interpretation is that a path goes through this artificial link if and only if it uses function f_k from node i. Altogether,

$$\begin{aligned}
\mathcal{N}_e(c) &= \bigcup_{k=1}^{K_c+1} \mathcal{N}_k(c)\\
|\mathcal{N}_e(c)| &= (K_c+1)N_R(c)\\
\mathcal{L}_e(c) &= \bigcup_{k=1}^{K_c+1} \mathcal{L}_k(c) \bigcup_{k=1}^{K_c} \hat{\mathcal{L}}_k(c)\\
|\mathcal{L}_e(c)| &= (K_c+1)L_R + \sum_{k=1}^{K_c} n_k(R,c)
\end{aligned}$$

where $n_k(R,c)$ is the number of nodes containing function f_k in the pruned network.

Remark 2: The first step of any solution that we propose will be the creation of the expanded network for each new connection request.

Remark 3: The expanded network for connection c is (K_c+1) times larger than the original network. The algorithms that we will design would have to be "fast" on the expanded network.

C. Notation

If we want to single out a node i in some $\mathcal{R}_k$, we use the notation (i,k). Similarly, a link in $\mathcal{R}_k$ is denoted by the pair $((i,k),(j,k))$ which we shorten to (i,j,k). A link in $\hat{\mathcal{L}}_k$ is denoted by (i,i,k). This is the artificial link from the function node i in $\mathcal{R}_k$ to the same node i in $\mathcal{R}_{k+1}$. It only exists if $i \in \mathcal{A}_k$ and $i \in \mathcal{N}_e$ since i would have been pruned if its remaining processing capacity is less than p.

We also define the following node sets to make the expression of the model easier. For each node $(i,k) \in \mathcal{N}_e$, we define the sets of incoming $\mathcal{I}$ and outgoing $\mathcal{O}$ links. Precisely, $\mathcal{I}(i,k)$ (resp. $\mathcal{O}(i,k)$) is the set of nodes j such that there is an incoming (resp. outgoing) link (j,i,k) (resp. (i,j,k)). Note that in these definitions, we can have $i=j$ whenever a link is an artificial link to or from node i in some set $\mathcal{R}_{k-1}$ or $\mathcal{R}_{k+1}$, as the case may be.

VI. Minimum-Cost Flow Model

Once we have created the expanded network for a new connection request, we need to find a path for that connection on that network. We model this problem as an integer program subject to capacity constraints in the expanded network. In the following, we call this problem the *snapshot*, if we want to emphasize that this problem is specific to a particular connection at some time instant, or the optimal *routing* problem to emphasize the optimal routing aspect.

A. Finding a Feasible Path

The first part of the model has to do with finding a path that will meet the capacity constraints. For this, we define $x_{i,j,k}$, the binary decision variables indicating that link $(i,j,k) \in \mathcal{L}_k$

is in the path. The problem can be stated as finding a set of $(x_{i,j,k})$ that meet the constraints

$$\sum_{j \in \mathcal{I}(i,k)} x_{j,i,k} = \sum_{j \in \mathcal{O}(i,k)} x_{i,j,k} + \delta_{i,k} \quad \forall (i,k) \in \mathcal{N}_e \tag{1}$$

$$\delta_{i,k} = \begin{cases} 1 & \text{if } i = o, k = 1 \\ -1 & \text{if } k = K+1, i = d \\ 0 & \text{otherwise} \end{cases}$$

$$b \sum_k x_{i,j,k} \le C_{i,j} \quad \forall (i,j) \in \mathcal{L}_k \tag{2}$$

$$p \sum_k \sum_{j \in \mathcal{I}(i,k)} x_{j,i,k} + \sum_k q_k \sum_{(i,i) \in \hat{\mathcal{L}}_k} x_{i,i,k} \le P_i \quad \forall i \in \mathcal{N} \tag{3}$$

$$x_{i,j,k} \in \{0,1\} \quad \forall (i,j,k) \in \mathcal{L}_e. \tag{4}$$

Equation (1) is the standard flow conservation equation at node (i,k). The term $\delta_{i,k}$ represents one unit of flow entering at o in copy 1 and leaving at node d in copy $K+1$. These equations insure that there will be one and only one path between node $(o,1)$ and $(d, K+1)$. The total capacity constraint on each link $(i,j) \in \mathcal{L}_k$ is given by (2). The processing constraint (3) is the total processing capacity to route a connection through the node plus the processing capacity required to execute f_k. In this model, we only need to find a set of feasible $x_{i,j,k}$ and we call this the *feasibility* problem.

B. Finding Good Paths

Of all the paths that satisfy (1–4), we would like to find the ones that produce low blocking. We can do this by adding an artificial length, or transportation cost, to the links, and converting the feasibility problem of Section VI-A into a minimum-cost flow problem. First, we define $a_{i,j,k} \ge 0$ as the cost of transporting one unit of flow on link (i,j,k). Note that these costs *do not* represent an actual transportation cost. Rather, we will use them in Section VII-C as a tool to lower the blocking. We will choose them differently for different heuristic solution algorithms. We then add an objective function Z to the feasibility problem so that the path finding model becomes the following *minimum-cost* flow problem.

$$\min_{\mathbf{x}} Z = \sum_k \sum_{(i,j) \in \mathcal{L}_k} a_{i,j,k} x_{i,j,k} + \sum_k \sum_{(i,i) \in \hat{\mathcal{L}}_k} a_{i,i,k} x_{i,i,k}$$
$$\text{subject to } (1-4). \tag{5}$$

Note that constraints (2) and (3) couple a number of links and nodes in the expanded network. This means that we cannot use the standard argument for capacitated minimum-cost flows that automatically have integer solutions for integer inputs. For this to be valid, the capacity constraints must apply to *each* link separately. This is why we need to state explicitly that the $x_{i,j,k}$'s are integer and why we must solve it with an integer programming algorithm. This makes the problem difficult and it is unlikely that we will be able to find optimal solutions for large networks within a time short enough to use in a virtual network environment. For this reason, we now focus on fast heuristic algorithms. Most of these are based on the Lagrangian Relaxation of (5).

C. Lagrangian Relaxation

Problem (5) is known as the minimum-cost flow with side constraints. It is particularly well suited for a solution by Lagrangian relaxation since the sub-problems are simple. We construct the partial Lagrangian function by relaxing constraints (2) and (3) with multipliers $\gamma_{i,j}, \mu_i \ge 0$. We get the Lagrange function

$$\begin{aligned}\mathcal{L}(\mathbf{x}, \boldsymbol{\gamma}, \boldsymbol{\mu}) &= \sum_k \sum_{(i,j) \in \mathcal{L}_k} (a_{i,j,k} + b\gamma_{i,j})\, x_{i,j,k} \\ &\quad + \sum_k \sum_{(i,i) \in \hat{\mathcal{L}}_k} a_{i,i,k} x_{i,i,k} \\ &\quad + p \sum_{i \in \mathcal{N}} \mu_i \sum_k \sum_{j \in \mathcal{I}(i,k)} x_{j,i,k} \\ &\quad + \sum_k \sum_{(i,i) \in \hat{\mathcal{L}}_k} \mu_i q_k x_{i,i,k} \\ &\quad - \left(b \sum_{(i,j) \in \mathcal{L}} \gamma_{i,j} C_{i,j} + \sum_{i \in \mathcal{N}} \mu_i P_i \right).\end{aligned}$$

The dual function Φ is defined at some feasible point $(\boldsymbol{\gamma}, \boldsymbol{\mu})$ as

$$\Phi(\boldsymbol{\gamma}, \boldsymbol{\mu}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\gamma}, \boldsymbol{\mu}) \tag{6}$$
$$\text{subject to } \ (1) \text{ and } (4). \tag{7}$$

The minimization over $\mathbf{x}$ is called the *sub-problem* for some value of $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$. The capacity constraints (2) and (3) are removed from the sub-problem and so it becomes an uncapacitated minimum-cost flow with the link lengths $w_{i,j,k}$ as

$$w_{i,j,k} = \begin{cases} a_{i,j,k} + b\gamma_{i,j} + p\mu_j & \text{if } i \ne j \\ a_{i,i,k} + q_k \mu_i & \text{if } i = j. \end{cases} \tag{8}$$

The first line of (8) is for the real links and the other, for the artificial links. Sub-problem (6)–(7) can be solved very quickly by any shortest-path algorithm, even for large expanded networks. This is the reason why the Lagrangian decomposition approach is so attractive.

Because of weak duality, we know that $\Phi(\boldsymbol{\gamma}, \boldsymbol{\mu}) \le Z(\mathbf{x})$ for any feasible set of vectors $\boldsymbol{\gamma}, \boldsymbol{\mu}$ and $\mathbf{x}$. We then try to maximize the value of Φ over the dual variables $\boldsymbol{\gamma}$ and $\boldsymbol{\mu}$ to improve that bound and hopefully get a primal feasible solution. The standard technique is the subgradient method where the $\boldsymbol{\gamma}$, $\boldsymbol{\mu}$ are modified iteratively at iteration s using:

$$\begin{aligned}\gamma_{i,j}^{(s+1)} &= \max\left\{0, \gamma_{i,j}^{(s)} + \alpha^{(s)} g_{i,j}^{(s)}\right\} \\ \mu_i^{(s+1)} &= \max\left\{0, \mu_i^{(s)} + \alpha^{(s)} h_i^{(s)}\right\}\end{aligned} \tag{9}$$

$$\begin{aligned}g_{i,j}^{(s)} &= b \sum_k x_{i,j,k}^{(s)} - C_{i,j} \\ h_i^{(s)} &= p \sum_k \sum_{j \in \mathcal{I}(i,k)} x_{j,i,k}^{(s)} + \sum_k \sum_{(i,i) \in \hat{\mathcal{L}}_k} q_k x_{i,i,k}^{(s)} - P_i\end{aligned} \tag{10}$$

where $\alpha^{(s)}$ is called the *step size* at iteration s. Simply stated, (9) says that one should increase the $\gamma_{i,j}$'s for links where the flow is larger than the capacity and decrease it if it is smaller. The same rule applies for (10) when the processing requirement exceeds the node capacity.

To summarize, we have reduced the initial problem to a sequence of shortest path problems on the expanded network. Given enough time, the sequence of subgradient iterations will yield an optimal solution to the dual that will be feasible for the primal. As we have said before, however, we need to compute solutions quickly so that iterative procedures such as this are not possible. For this reason, we now look at heuristic algorithms based on a few iterations.

VII. Heuristic Algorithms

We are now proposing different algorithms to find a feasible solution in a small amount of time. The first one is a greedy algorithm, called *Greedy Forward* in the following, that works on the original network. The other algorithms work on the expanded network. One is based on a single iteration of the subgradient method and the other uses two iterations. In both cases, different cost functions are used.

Note that none of these algorithms is guaranteed to meet the capacity constraints (2) and (3) since these have been replaced by a cost term (8) in the objective. When these algorithms terminate, the constraints are checked and if they are not met, the connection is lost. All these heuristics use a standard shortest path algorithm such as *Dijkstra* [23], *Bellman-Ford* [24], *A* search* [25], each with many variants. In this paper, we use Dijkstra's original variant given in [23] to find the shortest path between two nodes.

A. Greedy Forward

The basic idea of the algorithm is to find a sequence of shortest paths linking the functions in the real graph. The Greedy Forward algorithm uses the Dijkstra algorithm and starts by finding the shortest path from the origin o to set of nodes $\mathcal{A}_1$ containing the first function f_1 required by the connection. Let i_1 be the selected function node. Then we find the shortest path from i_1 to set of nodes $\mathcal{A}_2$ containing the second function f_2 required by the connection. Let i_2 be the selected function node. Then, for adjacent functions $(f_j, f_{j+1}) \in \mathcal{F}_c$, it finds the shortest path from i_j to set of nodes $\mathcal{A}_{j+1}$ containing function f_{j+1}. Finally, it finds the shortest path from i_K containing function f_K to the destination d of the connection.

Remark 4: The complexity of this algorithm is $K+1$ times that of computing a shortest path from some node I_k to the n_k function nodes in $\mathcal{A}_{k+1}$ on a N-node network.

B. The Subgradient Method: Choosing the Costs

Recall from Section VI-B that the link costs can be chosen to reduce network blocking. Because paths that use a larger number of links and nodes often produce large blocking, we may simply choose all the $a_{i,j,k} = 1$. This is equivalent to computing a *min-hop* path in the expanded network. This does not consider the link and node utilization and a second option is to choose values for the link costs $a_{i,j,k}$s to be large when they are close to their limit, thus leaving some resources available for later connections. We propose the following costs:

$$a_{i,j,k} = \begin{cases} \dfrac{b}{C_{i,j}} & \text{if } i \neq j \\ \dfrac{q_k}{P_i} & \text{if } i = j. \end{cases} \tag{11}$$

Remark 5: The costs defined here correspond to two families of algorithms. We will use the term *min-hop* when we use $a_{i,j,k} = 1$, and *shortest path with unequal costs* when we use (11). If we want to talk about both methods as one group, we use the plural form *shortest paths*.

C. Subgradient Method: A Single Iteration

First build the expanded network and set $\boldsymbol{\gamma} = \boldsymbol{\mu} = 0$ which is equivalent to ignoring the capacity constraints. The link length is thus

$$w_{i,j,k} = a_{i,j,k} \quad \forall (i,j,k) \in \mathcal{L}_e.$$

The objective (5) is now simply the sum of the flow costs on all links.

We first compute the min-cost path(s) from o to d. It may turn out that there is no such path because of the pruning and we say, in that case, that the connection has experienced a *blocking of type 1*. Note that, in that case, nothing can be done, there simply does not exist a path for the connection.

If we find some solutions for the unconstrained problem, we check if the capacity constraints (2–3) are met, in which case we say that the solution is feasible.

1) If there are feasible solutions, we consider two cases.
   a) If there is only one, this is the optimal path.
   b) If there are more than one, we choose one at random.
2) If there is no feasible solution, we say that the connection has experienced a *blocking of type 2*.

At this point, we may simply stop and declare that the connection is lost. The other option, if time permits, is to make one more iteration and try to find a feasible solution.

Remark 6: This method will yield solutions that are different depending on the way we choose the costs. We have tried different costs and found that the costs given in (11) gave the best results, i.e., the lowest blocking, as we shall see in Section VIII.

D. Subgradient Method: Two Iterations

When we have a type 2 blocking, we try one additional iteration of the *subgradient* algorithm to try to find a feasible path.

In a complete subgradient procedure, we would make many iterations and the lengths would eventually change enough that the current path will no longer be the shortest one and the algorithm will produce a new solution. Because of the time limit, however, we are limited to only one iteration so that we cannot go through this gradual adjustment process. This is why we want to choose a step size large enough that the link lengths of the current shortest path will increase by an amount large enough that it will no longer be the optimal solution.

First consider some path l that does not satisfy some capacity constraints after the first iteration. Define J_l and $\hat{J}_l$ as the length of path l before and after changing the multipliers. We can compute

$$\begin{aligned} J_l &= \sum_{(i,j,k)\in l} a_{i,j,k}x_{i,j,k} \\ \hat{J}_l &= \sum_{(i,j,k)\in l} a_{i,j,k}x_{i,j,k} \\ &\quad + \sum_{(i,j,k)\in l}(b\gamma'_{i,j}+p\mu'_j)x_{i,j,k} + \sum_{(i,i,k)\in l} q_k\mu'_i x_{i,i,k} \end{aligned}$$

where $\boldsymbol{\gamma}'$ and $\boldsymbol{\mu}'$ are the updated values from (9–10). The difference of path length when changing the multipliers is thus

$$\hat{J}_l - J_l = \sum_{(i,j,k)\in l}(b\gamma'_{i,j}+p\mu'_j)x_{i,j,k} + \sum_{(i,i,k)\in l} q_k\mu'_i x_{i,i,k}. \tag{12}$$

From (9–10), we get

$$\begin{aligned} \hat{J}_l - J_l &= \alpha\left(\sum_{(i,j,k)\in l}(bg_{i,j}+ph_j)x_{i,j,k} \right. \\ &\qquad \left. + \sum_{(i,i,k)\in l} q_k h_i x_{i,i,k}\right) \\ &= \alpha S. \end{aligned}$$

Note that in general, some, but not all, g and h can be negative so that the slope S could be negative. In this case, we cannot increase the length of path l. We can avoid this problem by fixing the length of the links with negative h or g to their current value and computing S only for those links where $g \geq 0$ and $h \geq 0$. In order to insure that a new path will be found after the update, we impose the condition that the difference of the lengths of the paths before and after changing the multipliers with step size α has to be greater than some value δ.

$$\hat{J}_l - J_l \geq \delta. \tag{13}$$

From this, we get the condition on α

$$\alpha \geq \frac{\delta}{S}. \tag{14}$$

Obviously, the bigger the value of δ, the more chances we have of making the current path non-optimal. We have tried several values of δ, i.e., $\{1, 2, 3, 5, 7, 10, 20, 25, 30\}$ to find the best parameter. We found out that $\delta = 10$ gives the best results.

VIII. Evaluation of the Heuristic Algorithms

All the numerical results were obtained on Intel Xeon X5690 processors running at 3.47GHz with 16 cores and 150GB RAM. All calculations were made on a single processor.

TABLE II
NETWORK LIMIT PARAMETERS

	P_i	$C_{i,j}$
Bandwidth-limited	4.00	1.14
Node-limited	1.71	2.66
Both	1.71	1.14

A. Network Parameters

The networks that we used are made up of 200 nodes, each with the same processing capacity P. We generate the links from a uniform random distribution with parameter π, the probability that there is a directed link between any given pair of nodes. All links have the same capacity C. We have tried the two values of P and C listed in Table II. We have not considered simpler topologies, like fat trees. If the algorithms work well in a network with a general topology, we feel that it would still be the case with trees or other simpler networks.

When we create the network, we define the set of functions $\mathcal{F}$ that all connections may use, say $f_1, f_2, \ldots f_F$. To simplify, we put a copy of each one of these F functions at random on n nodes. In the following, we will use $F = 10$ for all networks and consider two cases $n = 5$ and $n = 20$.

For a given network, when a connection arrives, it requires $K \leq F$ functions. The actual functions needed by the connection are chosen randomly from the F available functions. Each connection requires $b = 0.1$ unit of bandwidth on each link it uses, $p = 0.05$ processing units when it goes through a node and $q = 0.1$ processing units in a node whenever a function it needs is executed on that node,

We adjust the link and node parameters to generate three kinds of networks. In *bandwidth-limited* networks, the node capacities are made relatively large so that the link capacities are the more constraining. Conversely, we define *node-limited* networks as networks where the node constraints are dominant. The final case is where both constraints are equally important. The values are summarized in Table II.

B. Simulator

The most important measure of the quality of a routing algorithm is the connection blocking probability. This is measured from the event-driven stochastic simulation of a set of typical networks. Our simulation is programmed in Java. In the following, all time units are in seconds unless otherwise noted.

We first generate a network for a given value of $\pi = 0.032$. We have used this network in almost all the figures since the results and trends for networks with other values of π were similar to those presented here. The only exceptions are in Figures 12, where we have simulated networks for different values of π to see the impact of connectivity on our results, and Figures 9 and 11, where we use a lower value of π in order to get at least a small amount of blocking on these figures.

Connection arrivals are generated using a Poisson process of parameter λ chosen such that the overall blocking probability is below 0.2. The connection holding time is generated from an exponential process with average $\tau = 10$. Once a connection is generated, its source and destination are generated uniformly at random from all the possible (o, d) pairs. The set

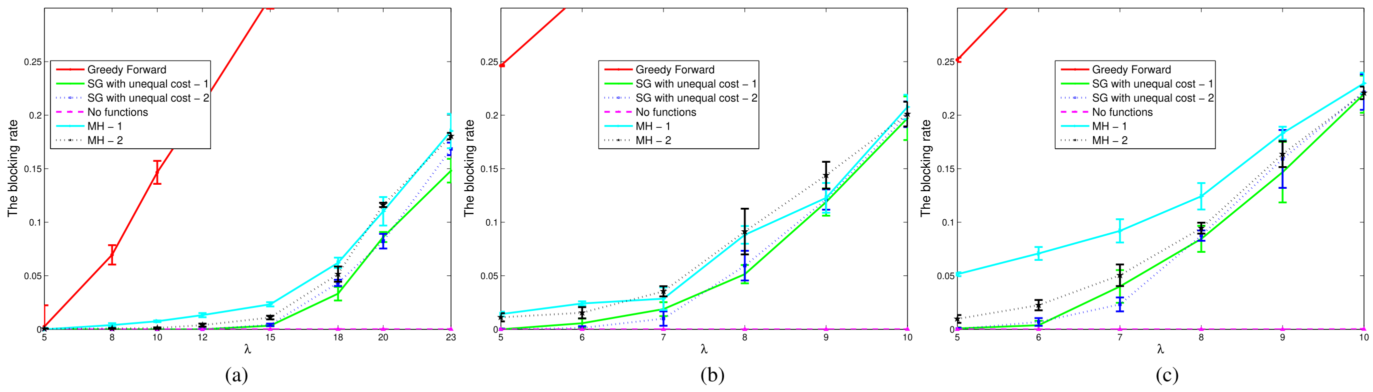


Fig. 4. Blocking vs λ, $n = 5$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

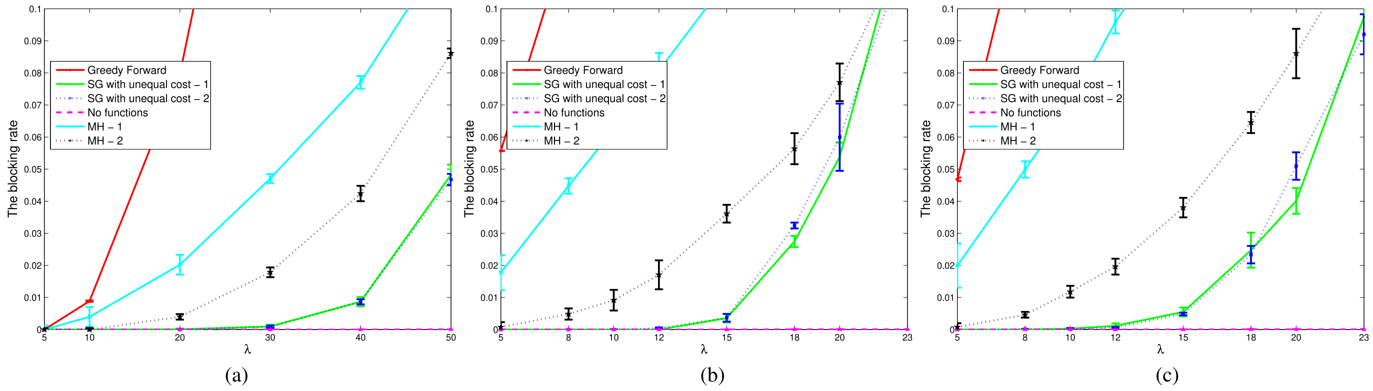


Fig. 5. Blocking vs λ, $n = 20$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

of K functions that the connection uses is selected randomly from $\mathcal{F}$. We generate 10 runs of length $T = 2000$ seconds each. The results are averaged out over the runs and we compute the 95% confidence interval for the results using a small-sample statistic.

C. Effect of Load

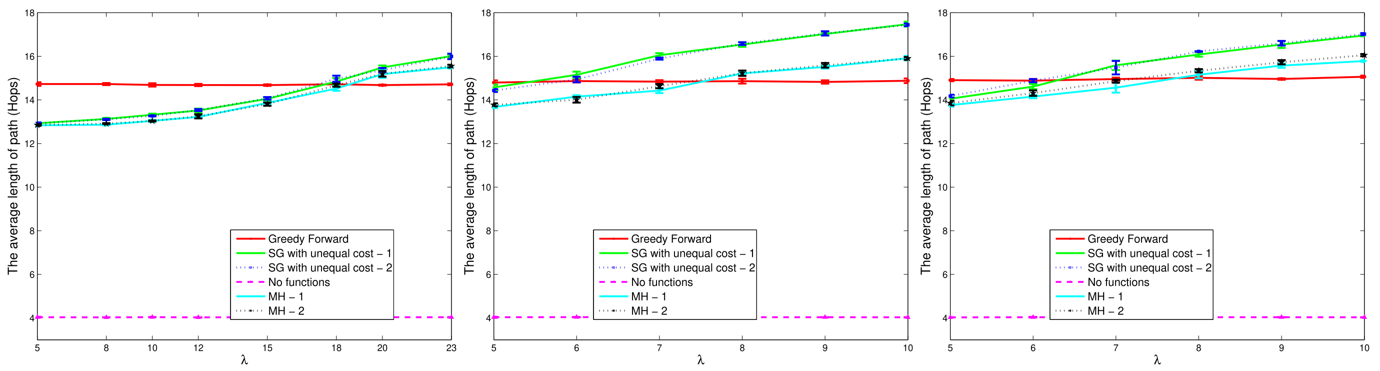
The first set of results shows the effect of λ on the blocking probability for the heuristic algorithms described in Section VII as well as the case where there is no function, where we compute a simple capacitated shortest path between o and d. We present in Figure 4 the case for $n = 5$ and in Figure 5, for $n = 20$ for the three kinds of networks. First, we see that the Greedy Forward algorithm produces significantly higher blocking than the subgradient-based algorithms. It is also remarkable that running the second iteration of the Lagrangian relaxation does not improve the performance of any one of the cases if we use the unequal costs to start with. There is, however, a significant improvement if we do the first iteration with unit costs. This finds a min-hop path, which is not very good, but the second iteration reduces the blocking because the updated link lengths computed from (9–10) now take into account the link and node loads.

We can see from these figures that the number n of copies of a function has a significant effect on the network blocking. If we choose a 5% level of blocking, we see that having 20 copies instead of 5 can more than double the amount of traffic that can be carried in the three kinds of networks using the two shortest path algorithms. This is directly tied to the length of paths produced by the algorithms as can be seen from Figures 6 and 7, where we have plotted the relative increment ρ in path length as a function of traffic. This is defined as $\rho = (L_i - L_0)/L_0$ where L_i is the average path length from algorithm i and L_0 is for the case where there is no function requirement. Note that here, the path length is the actual number of hops of the path, not the length as defined in the objective function. We can see that the algorithms finds shorter paths in the case $n = 20$. The reason is that when n is larger, we generate the networks with more function nodes so that the segments between two consecutive functions would tend to be shorter.

D. Effect of Costs

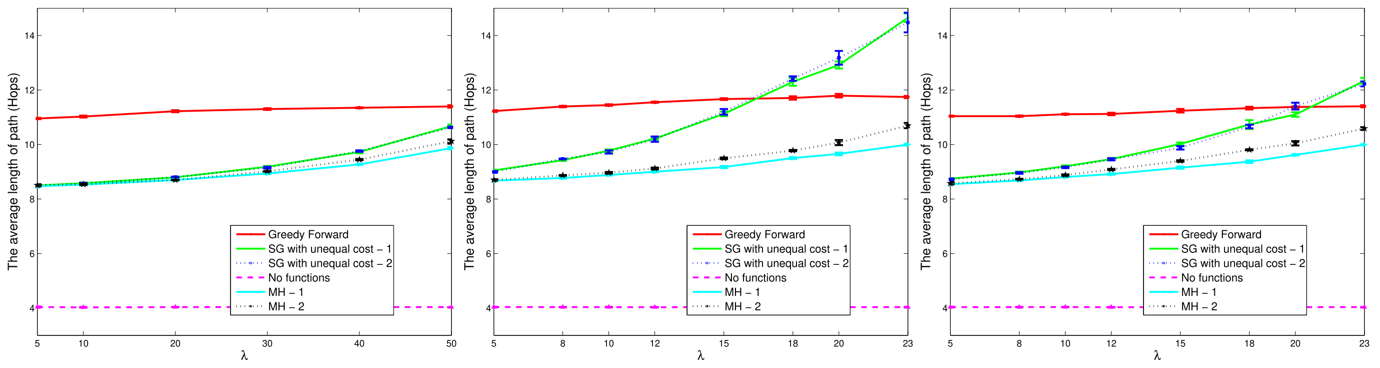
We can also see on Figures 4 and 5 the effect of the costs $a_{i,j,k}$ on the blocking. Using $a_{i,j,k} = 1$, i.e., a min-hop solution, produces a higher blocking than that produced when using the values of (11). This is an important result since min-hop is often chosen as the "best" routing option in many studies of the routing problem.

The results of Figures 6 and 7 show that even though the paths produced using the non uniform costs have more hops than the minimum possible, and sometimes by a significant amount, avoiding congested links is profitable overall in reducing the blocking.



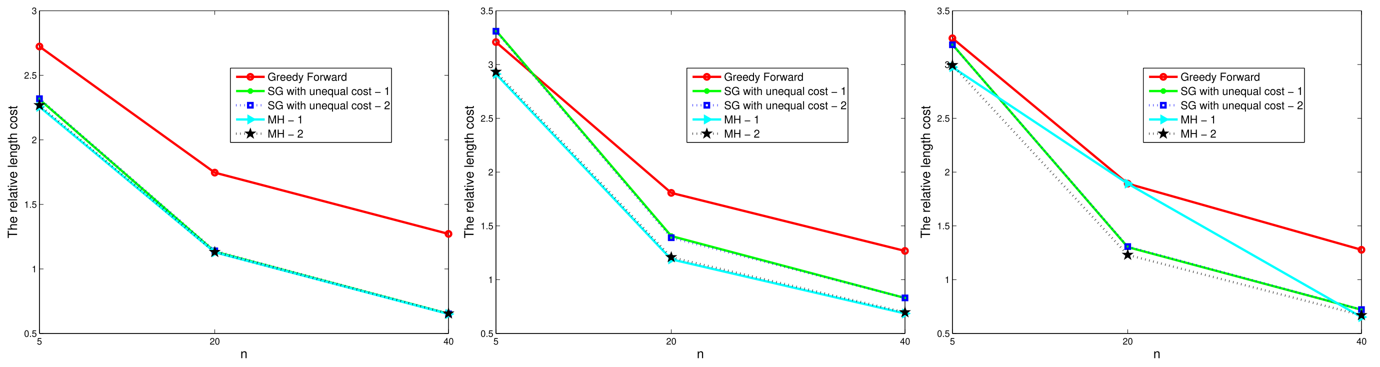
(a) (b) (c)

Fig. 6. Average path length vs λ, $n = 5$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.



(a) (b) (c)

Fig. 7. Average path length vs λ, $n = 20$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.



(a) (b) (c)

Fig. 8. Relative path length vs n, $\lambda = 10$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

E. Effect of the Number of Copies

We investigate next the effect of n, the number of copies of a function, by plotting on Figure 8 the average relative path length ρ as a function of n. It is quite clear that a large value of n will significantly reduce the path length, typically from a factor larger than 300% for $n = 5$ down to about 50% for $n = 40$.

F. Effect of K

We can also see from Figures 9 and 11 the effect that the number of functions K needed by each connection has on the blocking and the path length of the best algorithm, namely, "SG with unequal cost - 1". This is shown for the costs given by (11). We can see that in some cases, increasing the number of functions can have a large impact on blocking, as in 9(b) and (c). This impact can be reduced significantly by increasing the number of copies, as shown in the figures. In all cases, however, there is a significant increase in the number of hops, as expected.

G. Effect of Connectivity

We have seen that increasing the number of function copies improves the blocking by reducing the path length needed by the connections. We can do this also by increasing the

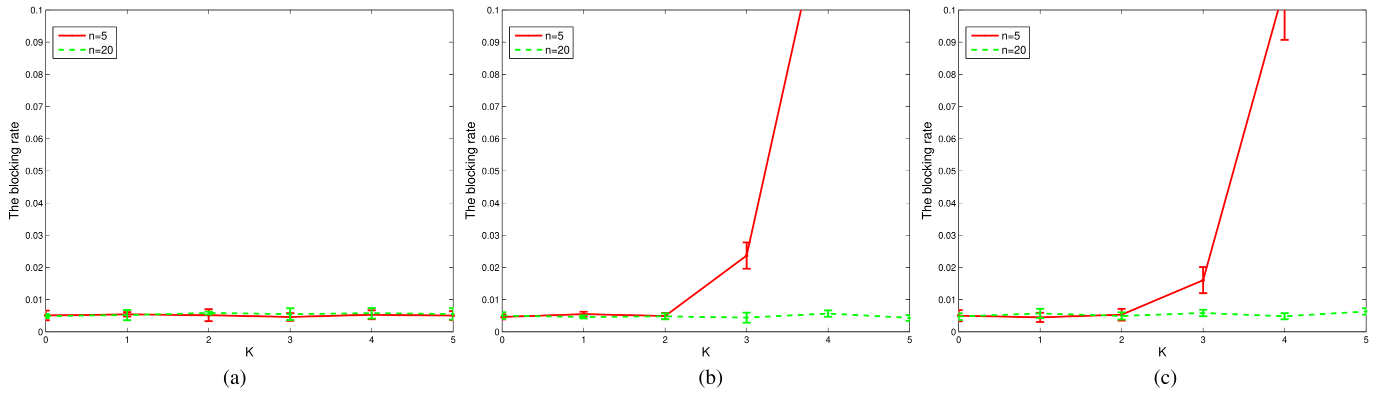


Fig. 9. Blocking of “SG with unequal cost - 1” vs K, $\lambda = 10$, $\pi = 0.027$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

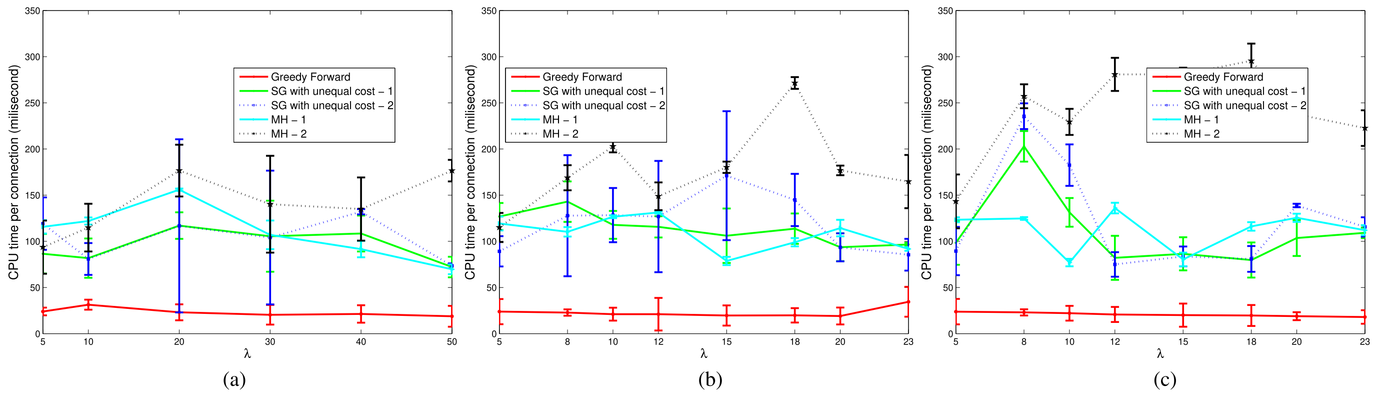


Fig. 10. Computation time vs λ, $n = 20$, $\pi = 0.032$, $K = 5$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

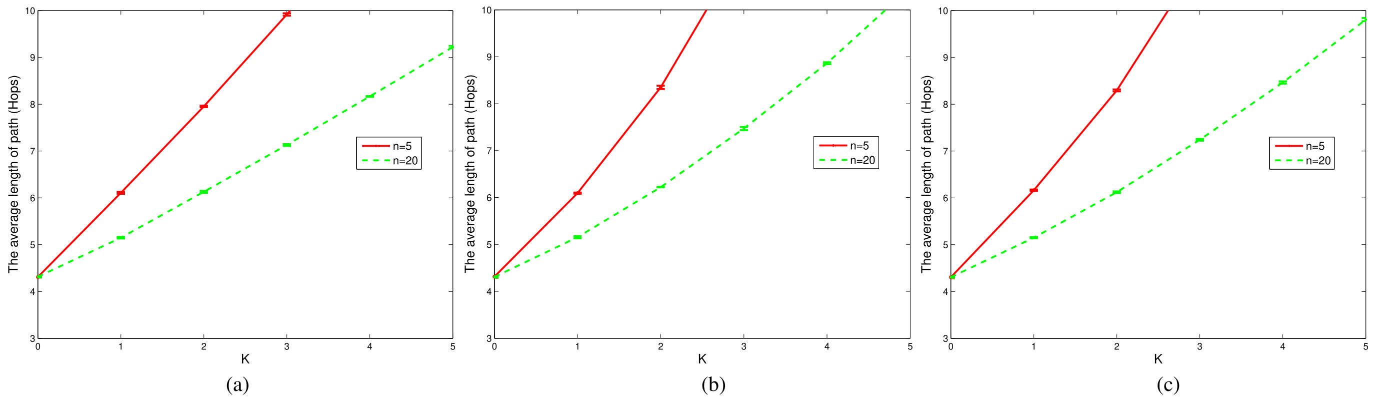


Fig. 11. Average path length of “SG with unequal cost - 1” vs K, $\lambda = 10$, $\pi = 0.027$. (a) Bandwidth limited. (b) Node limited. (c) Both limited.

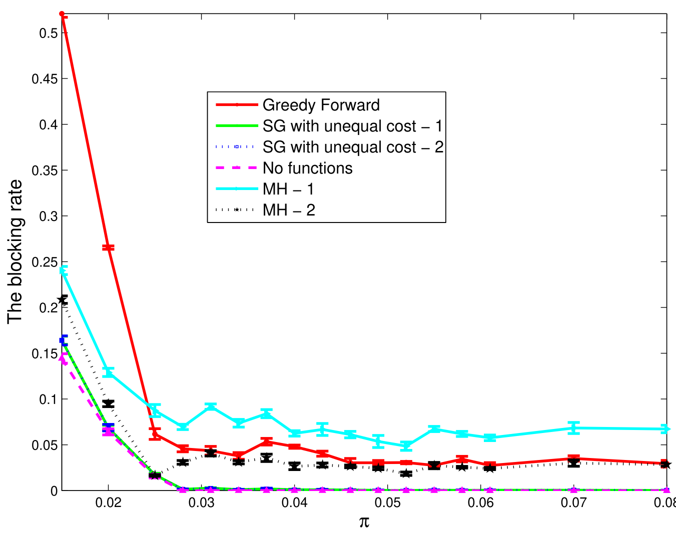
connectivity of the network. We look at this effect, as measured by π, on the performance. First, we see on Figure 12.(a) that increasing the number of links has a very significant effect on the blocking. If we double the number of links, the subgradient algorithm yields the same blocking as if there were no function requirements. This is due to a corresponding decrease in the average path length, as seen on Figure 12.(b).

H. Computation Time

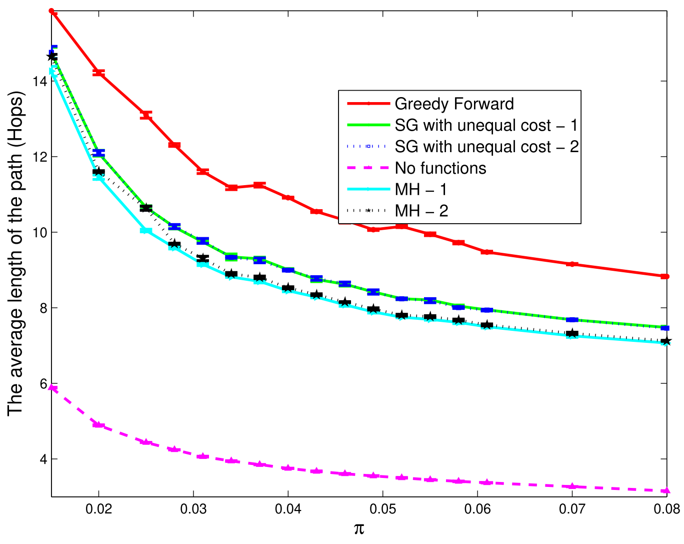
As discussed above, a path needs to be computed quite fast if we want to be able to serve the expected traffic in a large network. We can see in Figure 10 that this requirement is well met by the subgradient algorithms, which have computation times under 300 ms. As we can expect, the Greedy Forward algorithm is faster than the other ones since it computes $K+1$ shortest paths on a network of size N while the other ones compute a single shortest path on a network of size KN. This speed comes at the expense of a much lower accuracy, as we will now see.

I. Accuracy of the Heuristics

To fully evaluate the heuristics, we should compare their results with the ones obtained using the optimal solutions. This cannot be done easily on the type of simulations performed



(a)



(b)

Fig. 12. Impact of π, $n = 20$, $C = 2$, $P = 3$, $\lambda = 25$, $K = 5$. (a) Blocking rate vs π. (b) Path length vs π.

TABLE III
EXACT SOLUTIONS VS HEURISTICS, HIGH BLOCKING

	MH1	MH2	SG1	SG2
Blocking heuristics	0.29	0.32	0.13	0.22
Blocking optimal	0.04	0.06	0.03	0.05
Average heuristic length	4.44	6.06	7.21	6.77
Average salvaged mh length	5.56	7.27	5.8	6.82
Average salvaged sg length	5.56	7.27	5.8	6.82

TABLE IV
EXACT SOLUTIONS VS HEURISTICS, LOW BLOCKING

	MH1	MH2	SG1	SG2
Blocking heur	0.12	0.037	0.005	0.013
Blocking opt	0.018	0.022	0	0.003
Average heuristic length	5	5.2	6.5	6.6
Average salvaged mh length	6.5	6.7	7	7
Average salvaged sg length	6.5	6.7	7	7

for the above figures because of the large computation time of the optimal problem. Instead, we use the following approach.

Recall that the heuristics are really the solution of the routing sub-problem of the Lagrangian decomposition of Section VI-C where the capacity constraints have been relaxed. Because of this relaxation, the heuristics can fail to find a path for the connection because they ignore the capacity constraints when doing the shortest path calculation. Note that this cannot be avoided unless we explicitly take the constraints into account, which takes much more time, as we will see in Table V. One may then ask whether the decision to block the connection is simply due to the unconstrained shortest path calculation or if it is really the case that there were no paths available. We can answer this question by solving the exact constrained routing problem of Section VI-A.

To do so for a given heuristic algorithm, we run a simulation and take a random sample of *snapshots*[2] of the current network state at different points in time and solve the problem for each such snapshot. The results are shown in Tables III, for a network with high average blocking and IV for a lower blocking. Note that we select different samples for each heuristic. In these tables, the labels are SG, for Sub-Gradient, when we use the unequal costs of (11) and MH, for Min-Hop, when we use unit costs. In both cases, the label 1 or 2 indicates whether we did one or two iterations of the subgradient procedure.

[2]We took 100 snapshots for Table III and 600 for Table IV.

The first row shows the fraction of snapshots for which the heuristic was unable to find a path. This is an estimate of the network blocking probability when using the heuristic. The second row is the fraction of snapshots for which the exact algorithm could not find a path. Here too, this estimates the network blocking probability if we had used the exact solution of the routing problem. Equivalently, this is the fraction of the time the heuristic was right to block the connection.

We can see that there is a substantial difference between the two rows when the blocking is high. In other words, there is still some room to spare even though the heuristic cannot find it. This effect is still present at lower blocking but it is much smaller in absolute terms although the reduction in blocking for MH1 is still one order of magnitude.

The fact that the exact solution can find a path does not mean that we should necessarily use it. The reason is that this path might be very long. If this were the case, we would tie up a large number of resources to connect a single connection and it might be more efficient to drop this particular connection so that we will be able to connect maybe two or more shorter connection later on.

This question is partly answered in lines 3 to 5. Line 3 shows the average length of the paths found by the heuristics. The next two lines show the average length of the so-called *salvaged* connections, i.e., the ones that were blocked by the heuristic but could have been connected if we had used the exact solution, either using the min-hop or the shortest path metric of (11). The important point is that

TABLE V
COMPUTATION TIME OF THE OPTIMAL SOLUTION (Sec)

Nodes	Funct	Max copies	ampl	Min-hop	Shortest Path
50	3	3	0.11	0.036	0.02
50	5	3	0.24	0.056	0.036
50	10	3	0.8	0.42	0.28
50	10	5	0.85	0.24	0.24
200	20	3	58.2	5.89	5.00
200	20	5	62.3	4.87	7.2
200	20	10	64	11.1	7.37
100	20	5	11.6	0.34	0.24
100	20	5	11.2	0.28	0.18

these values are not very different from the path lengths found by the heuristics, sometimes a bit larger, sometimes smaller. In other words, the salvaged paths are not significantly longer than the other paths so that there may be a definite advantage in using them. This could be done if we used the exact algorithm in the simulation. This of course raises the question of the computation time that would be required. We can get some insight from the results shown in Table V. Except for the last two, all lines correspond to a network generated artificially. For each network, the number of nodes is indicated in column 1. The set of links is generated randomly according to the Barabasi-Albert distribution [26], which tends to produce networks with a topology similar to real networks. The number of functions used is shown in column 2 "Funct". For each function, the number of copies actually used is generated randomly up to a maximum value indicated in column 3 "Max copies" as well as the nodes where these copies will be installed. In other words, each function may have a different number of copies and these are installed at different nodes.

The last two lines, on the other hand, corresponds to the processing time averaged over all the network snapshots used in producing Tables III and IV. The main difference is that for the snapshots, *all* the available functions are used instead of some random subset, as in the previous lines.

For each network, we solve the exact routing problem (5) to optimality. This is done with Cplex called from an ampl model. In each case, the total processing time is divided into two parts. The first one, in column "ampl", is the time required for the ampl pre-processor to convert the model into a form acceptable to Cplex. The last two columns represent the actual processing time needed by Cplex to compute a solution. The column "Min-hop" is for results when we use the min-hop link lengths $a_{i,j,k} = 1$ while the column "Shortest path" is when we use the link lengths given by (11).

We can see that the ampl pre-processing takes much longer than the actual solution time of the problem by Cplex. Even then, the computation time varies between about 5 and 10 seconds for networks of a few hundred nodes. This means that using the exact solution of the routing problem in the simulation would require a very large computation time. It is not clear if such computation times are feasible in a real-time environment and if so, whether the increase in computation time will produce a reduction in blocking worth the effort.

IX. CONCLUSIONS AND FUTURE WORK

We have proposed two classes of algorithms based on the construction of an expanded network to take into account the function constraints and the Lagrangian decomposition [27] of the resulting problem to compute approximate solutions. We have then used simulation to provide extensive results on their performance. These results provide us with some interesting insights both about the algorithms and the network performance.

As far as the algorithms are concerned, we can see that

1) It is quite clear that the Greedy Forward algorithm should be used with care. It is significantly faster than the other algorithms presented here but also yields a significantly higher blocking.
2) Simply minimizing the number of hops is not the best strategy.
3) The cost functions should reflect the network congestion.
4) Doing a second subgradient iteration can reduce the blocking significantly if we use a min-hop algorithm. The improvement is negligible when we use the metric of (11).
5) All these unconstrained shortest path techniques may be unable to detect a number of free paths depending on the network congestion.

As far as network performance is concerned,

1) When connections require service chains, blocking becomes much worse.
2) Shortening the paths helps which can be done by having more copies of the functions or by increasing the number of links (which can be easily done in virtual networks).

This work can be extended in many directions. One such direction is to study the relative impact of connections with different requirements. Typically, connections with large requirements are harder to fit in a network. This means that they could experience a large blocking unless the algorithms are modified to take this into account.

It is also likely that the placement of functions will play a significant role since this will impact the path length needed to meet the requirements. The obvious solution is to place copies everywhere but there may be technical or cost constraints, e.g., if a function needs access to a very large database that cannot be cheaply replicated. The question of the optimal placement of functions will have to be examined.

The approach taken here is to manage the routing of connections. Once a connection is in place, its flow is constrained to follow a single path. Load management is done automatically when further connections arrive that can use spare capacity. This is quite different from other techniques mentioned in Section II that manage the packet traffic by splitting the flows, creating new copies and rearranging the network topology.

Comparing these techniques with each other will be difficult since they use different performance metrics. Connection-based approaches, like the one discussed here, try to minimize connection blocking and assume that the bandwidth and processing requirements are large enough to yield a good quality of service at the packet level.

The packet-level algorithms use different measures such as average packet delay and these are monitored in real time so that the controller can react to statistical changes in the traffic generated by a flow, which is a very different mode of operation. The only basis that might be common to both could be the network throughput at the same level of quality, taking into account the processing requirement of each technique. This is a major undertaking that is left for future work.

References

[1] ETSI. (Oct. 2012). *Network Functions Virtualization—Introductory White Paper*. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] X. Li and C. Qian, "A survey of network function placement," in *Proc. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2016, pp. 948–953.

[3] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[4] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proc. Int. Colloq. Struct. Inf. Commun. Complex. (SIROCCO)*, 2015, pp. 104–118.

[5] L. E. Li *et al.*, "PACE: Policy-aware application cloud embedding," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 638–646.

[6] J. Elias, F. Martignon, S. Paris, and J. Wang, "Efficient orchestration mechanisms for congestion mitigation in NFV: Models and algorithms," *IEEE Trans. Serv. Comput.*, vol. 10, no. 4, pp. 534–546, Jul./Aug. 2017.

[7] M. Obadia *et al.*, "Revisiting NFV orchestration with routing games," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 107–113.

[8] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–6.

[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2015, pp. 171–177.

[10] W. Ma, C. Medina, and D. Pan, "Traffic-aware placement of NFV middleboxes," in *Proc. IEEE GLOBECOM*, Dec. 2015, pp. 1–6.

[11] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Dynamic traffic steering of multi-tenant virtualized network functions in SDN enabled data centers," in *Proc. IEEE 21st Int. Workshop Comput. Aided Modelling Design Commun. Links Netw. (CAMAD)*, Oct. 2016, pp. 65–70.

[12] T.-M. Nguyen, S. Fdida, and T.-M. Pham, "A comprehensive resource management and placement for network function virtualization," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Bologna, Italy, Jul. 2017, pp. 1–9.

[13] M. Ghaznavi *et al.*, "Elastic virtual network function placement," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2015, pp. 255–260.

[14] T. Kuo, B. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[15] S. Palkar *et al.*, "E2: A framework for NFV applications," in *Proc. 25th Symp. Operating Syst. Princ. (SOSP)*. New York, NY, USA: ACM, 2015, pp. 121–136. doi: 10.1145/2815400.2815423.

[16] A. Gember *et al.* (2013). "Stratos: A network-aware orchestration layer for virtual Middleboxes in clouds." [Online]. Available: https://arxiv.org/abs/1305.0209

[17] G. Even, M. Medina, and B. Patt-Shamir, "Competitive path computation and function placement in SDNs," in *Proc. 18th Int. Symp. Stabilization, Saf., Secur. Distrib. Syst.*, Nov. 2016, pp. 131–147.

[18] A. Dwaraki and T. Wolf, "Adaptive service-chain routing for virtual network functions in software-defined networks," in *Proc. Workshop Hot Topics Middleboxes Netw. Function Virtualization (HotMIddlebox)*. New York, NY, USA: ACM, 2016, pp. 32–37. [Online]. Available: http://doi.acm.org/2940147.2940148

[19] A. M. Medhat *et al.*, "Near optimal service function path instantiation in a multi-datacenter environment," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 336–341.

[20] S. Bhat and G. N. Rouskas, "Service-concatenation routing with applications to network functions virtualization," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul./Aug. 2017, pp. 1–9.

[21] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in SDNs," in *Proc. Int. Colloq. Struct. Inf. Commun. Complex. (SIROCCO)*, Jul. 2016, pp. 374–390.

[22] F. Kelly, *Notes on Effective Bandwidths* (Royal Statistical Society Lecture Notes Series), vol. 4. New York, NY, USA: Oxford Univ. Press, 1995, ch. 8, pp. 141–168.

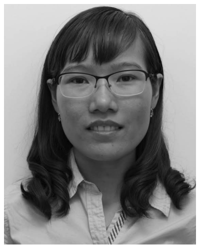
[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[24] R. Bellman, "On a routing problem," in *Quarterly of Applied Mathematics*. Providence, RI, USA: American Mathematical Society, 1958, pp. 87–90.

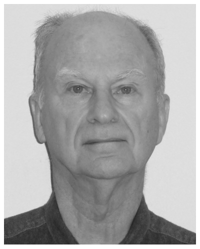
[25] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[26] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, pp. 47–97, 2002.

[27] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA, USA: Addison-Wesley, 1984.

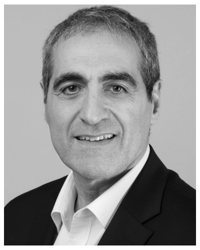


Thi-Minh Nguyen received the B.A. and M.A degrees in computer science from the Hanoi University of Science and Technology, Vietnam, in 2013, and the Ph.D. degree in networks and performance analysis from Paris 6, France, in 2017. She was a Lecturer with the National University of Civil Engineering, Vietnam. After one year of a post-doctoral position with Paris 6, she moved to industry, where she is currently a Research Engineer. Her research focused on the optimization problem in NFV Infrastructure. She is particularly interested in mathematics and graph theory.



André Girard received the Ph.D. degree in physics from the University of Pennsylvania, Philadelphia, PA, USA, in 1971. He was a Faculty Member at INRS-Telecommunications until 2004. During that time, he involved in various aspects of telecommunication networks, and in particular with performance evaluation, routing, dimensioning, and reliability. He has made numerous theoretical and algorithmic contributions to the design of telephone, ATM, IP, and wireless networks. He was an Adjunct Professor with INRS-EMT and École Polytechnique of Montréal, QC, Canada.

Catherine Rosenberg (M'89–SM'96–F'11) is currently a Professor and Canadian Research Chair in the future Internet with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. She is also the Cisco Research Chair in 5G Systems. Her research interests include networking, wireless, and energy systems. She is a Fellow of the Canadian Academy of Engineering.



Serge Fdida (SM'98) has been a Professor with Sorbonne Université (formally UPMC) since 1995. He has published numerous scientific papers, in addition to a few patents and one RFC. His research interests include the future Internet technology and architecture. He has been leading many research projects in Future Networking in France and Europe, notably pioneering the European activity on federated Internet testbeds. He is currently leading the Equipex FIT, a large-scale testbed on the Future Internet of Things. He is a Distinguished ACM Member. He has also developed a strong experience related to innovation and industry transfer. He was the Co-Founder of the Qosmos company—one of the active contributors to the creation of the Cap Digital cluster in Paris. He held various community and management responsibilities in various organizations including Sorbonne Université.