Approximate and Incremental Network Function Placement

Tamás Lukovszki[a], Matthias Rost[b], Stefan Schmid[c]

[a] *Eötvös Loránd University, Hungary*
[b] *Technische Universität Berlin, Germany*
[c] *Aalborg University, Denmark*

Abstract

The virtualization of modern computer networks introduces interesting new opportunities for a more flexible placement of network functions and middleboxes (firewalls, proxies, traffic optimizers, virtual switches, etc.). This paper studies approximation algorithms for the incremental deployment of a minimum number of middleboxes, such that capacity constraints at the middleboxes and length constraints on the communication routes are respected.

Based on a new, purely combinatorial and rigorous proof of submodularity, we obtain our main result: a deterministic greedy approximation algorithm which only employs augmenting paths to serve future communication pairs. Hence, our algorithm does not require any changes to the locations of existing middleboxes or the preemption of previously served communication pairs when additional middleboxes are deployed. It is hence particularly attractive for incremental deployments. We prove that the achieved polynomial-time approximation bound is optimal, unless $P = NP$ holds.

This paper also initiates the study of a weighted problem variant, in which entire groups of nodes need to communicate via a middlebox, possibly at different rates. We present an LP relaxation and randomized rounding algorithm for this problem, leveraging an interesting connection to scheduling.

We complement our formal results with a simulation study of a large set of synthetically generated instances. Our results indicate that the presented algorithms yield near-optimal solutions in practice.

Email addresses: lukovszki@inf.elte.hu (Tamás Lukovszki), mrost@inet.tu-berlin.de (Matthias Rost), schmiste@cs.aau.dk (Stefan Schmid)

1. Introduction

Middleboxes are ubiquitous in modern computer networks and provide a wide spectrum of in-network functions related to security, performance, and policy compliance. It has recently been reported that the number of middleboxes in enterprise networks can be of the same order of magnitude as the number of routers [39].

While in-network functions were traditionally implemented in specialized hardware appliances and middleboxes, computer networks in general and middleboxes in particular become more and more software-defined and virtualized [18]: network functions can be implemented in software and deployed fast and flexibly on the virtualized network nodes, e.g., running in a virtual machine on a commodity x86 server. Modern computer networks also offer new flexibilities in terms of how traffic can be *routed* through middleboxes and virtualized data plane appliances (often called Virtual Network Functions, short *VNFs*) [36]. In particular, the advent of *Software-Defined Network (SDN)* technology allows network operators to steer traffic through middleboxes (or chains of middleboxes) using arbitrary routes, i.e., along routes which are not necessarily shortest paths, or not even loop-free [3, 34, 13, 33]. In fact, OpenFlow, the standard SDN protocol today, not only introduces a more flexible routing, but itself allows to implement basic middlebox functionality, on the switches [14]: an OpenFlow switch can match, and perform actions upon, not only layer-2, but also layer-3 and layer-4 header fields.

However, not much is known today about how to exploit these flexibilities algorithmically. A particularly interesting problem regards the question of where to deploy a minimum number of middleboxes such that basic routing and capacity constraints are fulfilled. Intuitively, the smaller the number of deployed network functions, the longer the routes via these functions, and a good tradeoff between deployment costs and additional latency must be found. Moreover, ideally, middleboxes should be *incrementally deployable*: when additional middleboxes are deployed, existing placements do not have to be changed. This is desirable especially in deployment scenarios with budget constraints, where an existing deployment has to be extended by new middleboxes.

1.1. Our Contributions

We initiate the study of the natural problem of (incrementally) placing a minimum number of middleboxes or network functions.

Our main technical result is a deterministic and greedy (polynomial-time) $O(\log(\min\{\kappa, n\}))$-approximation algorithm for the (incremental) middlebox placement problem in n-node networks where capacities are bounded by κ. The algorithm is attractive for incremental deployments: it does not require changes to the locations of existing middleboxes or the preemption of previously served communication pairs when additional middleboxes are deployed.

At the heart of our algorithm lies a new and purely combinatorial proof of the submodularity of the function maximizing the number of pairs that can be served by a given set of middleboxes. The submodularity proof directly implies a deterministic approximation algorithm for the minimum middlebox deployment problem. We show that the derived approximation bound is asympotically optimal, unless $P = NP$.

We believe that our model and approach has ramifications beyond middlebox deployment. For instance, our model also captures fundamental problems arising in the context of incremental SDN deployment (solving an open problem in [9, 29]) or distributed cloud computing where resources need to be allocated for large user groups.

We also initiate the study of a weighted problem variant, where entire groups of nodes need to communicate via a shared network function (e.g., a multiplexer of a multi-media conference application or a shared object), possibly at different rates. Based on an interesting connection to scheduling [23], we obtain a greedy approximation algorithm for this problem as well, while losing the incremental deployment property.

We also investigate the performance of both algorithms in an extensive computational evaluation. In particular, we study the empirical approximation factors for both approximation algorithms and their runtime. We also study the incremental deployment property in detail and show that greedily deploying the middleboxes initially allows to consistently serve nearly as many communication requests as when optimally deploying middleboxes; only when the number of deployed middleboxes approaches the number of required middleboxes in the optimal solution, the ability to arbitrarily (re-)place middleboxes pays off. In order to ensure reproducability as well as to ease future research, we have published our implementation as open source [37].

1.2. Novelty and Related Work

In this paper we are interested in algorithms which provide formal approximation guarantees. In contrast to classic covering problems [8, 15, 41]:

(1) we are interested in the distance *between communicating pairs*, *via* the covering nodes, and not *to* the covering nodes; (2) we aim to support *incremental deployments*: middlebox locations selected earlier in time as well as the supported communication pairs should not have to be changed when deploying additional middleboxes; (3) we consider a *capacitated* setting where the number of items which can be assigned to a node is bounded by κ.

To the best of our knowledge, so far, only heuristics or informal studies without complete or combinatorial proofs of the approximability [2, 25, 30, 39] as well as algorithms with an exponential runtime in the worst-case [20, 25], of the (incremental and non-incremental) middlebox deployment problem have been presented in the literature. Our work also differs from function chain embedding problems [12, 27, 11] which often revolve around other objectives such as maximum request admission or minimum link load.

Nevertheless, we in this paper show that we can build upon hardness results on uncapacitated covering problems [28] as well as Wolsey's study of vertex and set covering problems with hard capacities [41]. An elegant alternative proof to Wolsey's dual fitting approach, based on combinatorial arguments, is due to Chuzhoy and Naor [7]. In [7] the authors also show that using LP-relaxation approaches is generally difficult, as the integrality gap of a natural linear program for the weighted and capacitated vertex and set covering problems is unbounded.

Bibliographic Note. A preliminary version of this paper appeared in CCR [26].

1.3. Putting Things into Perspective

Our problem is a natural one and, as mentioned, features some interesting connections to and has implications for classic (capacitated) problems. To make things more clear and put things into perspective, in the following, we will elaborate more on this relationship.

The generalized dominating set problem [4] asks for a set of dominating nodes of minimal cardinality, such that the distance from any network node to a dominator is at most ℓ. In the capacitated version of the problem, the number of network nodes which can be dominated by a node is limited. Similarly, capacitated facility location problems [1, 5] ask for locations to deploy facilities, subject to capacity constraints, such that the number of facilities as well as the distance to the facilities are jointly optimized. In contrast to these problems where the distance *to* a dominator or facility *from a given*

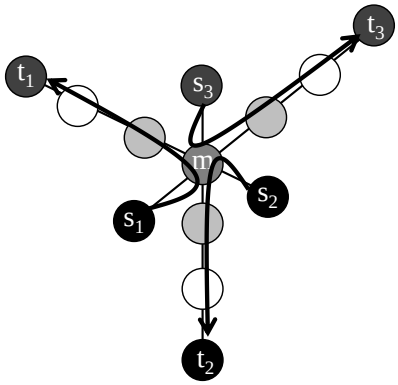

Figure 1: Example: A minimum cardinality dominating set (three nodes, in *light grey*) is a bad approximation for the middlebox deployment problem (one node in center, in *darker grey*).

node is measured, in our middlebox deployment problem, we are interested in the distance (resp. stretch) *between node pairs*, *via* the middlebox. At first sight, one may intuitively expect that optimal solutions to dominating set or facility location problems are also good approximations for the middlebox deployment problem. However, this is not the case, as we illustrate in the following. Consider an optimal solution to the distance $\ell/2$ dominating set problem: since the distance to any dominator is at most $\ell/2$, the locations of the dominators are also feasible locations for the middleboxes: the route between two nodes via the dominator is at most $\ell/2 + \ell/2 = \ell$. However, the number of dominators in this solution can be much larger than the number of required middleboxes.

Consider the example in Figure 1: in a star network where communicating node pairs are located at the leaves, at depth 1 and $\ell - 1$, and where node capacities are sufficiently high such that they do not constitute a bottleneck, a *single* middlebox m in the center is sufficient. However, the stricter requirement that dominating nodes must be at distance at most $\ell/2$, results in a dominating set of cardinality $\Omega(n/\ell)$, i.e., $\Omega(n)$ for constant ℓ.

While our discussion revolved around constraints on the length ℓ, similar arguments and bounds also apply to the *stretch*, the main focus of this paper: the ratio of the length of the path through a middlebox, and the length of the shortest path between the communicating pair. To see this, simply modify the example in Figure 1 by adding a direct edge between each communicating pair s_i and t_i. Let $c = \ell$. Then the length of the shortest path between each

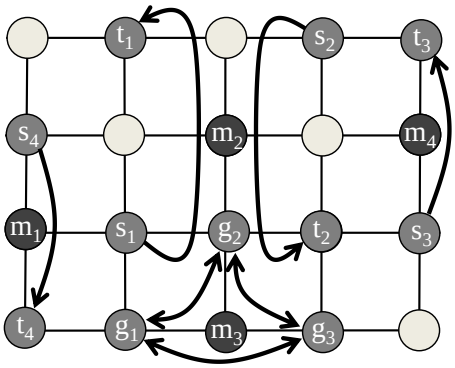

Figure 2: Example: The communication between node pairs (s_i, t_i), for $i \in \{1, 2, 3, 4\}$, as well as between the group of nodes $\{g_1, g_2, g_3\}$ needs to be routed via a network function resp. middlebox M. Due to capacity constraints and constraints on the route length, M is instantiated at four locations $\{m_1, m_2, m_3, m_4\}$ in this example.

pair is one. The dominating nodes must be at distance at most $c/2$ from the communicating nodes. Consequently, the cardinality of the dominating set is $\Omega(n/c)$, i.e., $\Omega(n)$ for constant c; deploying one single middlebox m at the center and routing the communication between each pair through m results in paths of stretch c.

1.4. Organization

The remainder of this paper is organized as follows. Section 2 introduces our model and discusses different use cases. In Section 3 we present our approximation algorithm together with its analysis. Section 4 extends our study to weighted and group models. After reporting on our simulation results in Section 5, we conclude our contribution in Section 6.

2. Model and Use Cases

2.1. Formal Model

We model the computer network as a graph connecting a set V of $n = |V|$ nodes. The input to our problem is a set of communicating node pairs P: the route of each node pair $(s, t) \in P$ needs to traverse an instance of a middlebox resp. network function (e.g., a firewall). Node pairs do not have to be disjoint: nodes can participate in many communications simultaneously.

For the sake of generality, we assume that middleboxes can only be installed on a subset of nodes $U \subseteq V$. We will refer to the set of middleboxes

locations (and equivalently, the set of middleboxes instances) by M, and we are interested in deploying a minimal number of middleboxes at legal locations $M \subseteq U$ such that:

1. Each pair $p = (s, t) \in P$ is assigned to an instance $m \in M$, denoted by $m = \mu(p)$.
2. For each pair $p = (s, t) \in P$, there is a route from s via $m = \mu(p)$ to t of length at most $\rho \cdot d(s, t)$, i.e., $d(s, m) + d(m, t) \leq \rho \cdot d(s, t)$, where $d(u, v)$ denotes the length of the shortest path between nodes $u, v \in V$ in the network G, and where $\rho \geq 1$ is called the *stretch*. For example, a stretch $\rho = 1.2$ implies that the distance (and the latency accordingly) when routing via a middlebox is at most 20% more than the distance (and the latency accordingly) when connecting directly. Our approach supports many alternative constraints, e.g., on the maximal route length.
3. Capacities are respected: at most κ node pairs can be served by any middlebox instance.

Our objective is to minimize the number of required middlebox instances, subject to the above constraints.

In this paper, we will also initiate the study of a weighted model, where different communication pairs have different demands, as well as a group model, where network functions need to serve entire groups of communication partners. See Figure 2 for an example.

2.2. Use Cases

Let us give three concrete examples motivating our formal model.

Middlebox Deployment. Our model is mainly motivated by the middlebox placement flexibilities introduced in network function virtualized and software-defined networks. Deploying additional middleboxes, network processors or so-called "universal nodes" can be costly, and a good tradeoff should be found between deployment cost and routing efficiency. For example, today, network policies can often be defined in terms of adjacency matrices or big switch abstractions, specifying which traffic is allowed between an ingress port s and an outgress network port t. In order to enforce such a policy, traffic from s to t needs to traverse a middlebox instance inspecting and classifying the flows. The location of every middlebox can be optimized, but is subject to the constraint that the route from s to t via the middlebox should not be much longer than the shortest path from s to t.

Deploying Hybrid Software-Defined Networks. There is a wide consensus that the transition of existing networks to SDN will not be instantaneous, and that SDN technology will be deployed incrementally, for cost reasons and to gain confidence. The incremental OpenFlow switch deployment problem can be solved using waypointing (routing flows via OpenFlow switches); our paper solves the algorithmic problem in the Panopticon [25] system.

Distributed Cloud Computing. The first two use cases discussed above come with per-pair requirements: each communicating pair must traverse at least one function. However, there are also scenarios where entire groups G_i of nodes need to share a waypoint: for example, a multiplexer of a group communication (a multi-media conference) or a shared object in a distributed system, e.g., a shared and collaborative editor: An example could be a distributed cloud application: imagine a set of users G_i who would like to use a collaborative editor application à la Google Docs. The application should be hosted on a server which is located close to the users, i.e., minimizing the latency between user pairs.

3. Approximation Algorithm

This section presents a deterministic and polynomial-time $O(\log(\min\{n, \kappa\}))$-approximation algorithm for the middlebox deployment problem. Our algorithm is based on an efficient computation of a certain submodular set function: it defines the maximum number of pairs which can be covered by a given set of middleboxes. In a nutshell, the submodular function is computed efficiently using an augmenting path method on a certain bipartite graph, which also determines the corresponding assignment of communication pairs to the middleboxes. The augmenting path algorithm is based on a simple, linear-time breadth-first graph traversal. The augmenting path method is attractive and may be of independent interest: similarly to the flow-based approaches in the literature [6], it does not require changes to previously deployed middleboxes, but removes the disadvantage of [6] that the set of served communication pairs changes over time: an attractive property for *incremental deployments*.

Our solution with augmenting paths results (theoretically and practically) in significantly faster computations of the submodular function value as well as of the corresponding assignment compared to the flow based approach.

In fact, computing all augmenting paths takes $O(|E| \min\{\sqrt{|V|}, \kappa\})$ by using the Hopcroft-Karp algorithm [21] performing at most $O(\min\{\sqrt{|V|}, \kappa\})$ breadth-first search and depth-first search traversals of the graph. Computing a maximum flow takes $O(|V| \cdot |E|^2)$ time by the Edmonds-Karp algorithm [10], $O(|V|^3)$ time by the push-relabel algorithm of Goldberg and Tarjan [19], and $O(|V| \cdot |E|)$ time by the algorithm of Orlin [31] (see [19] for an overview on efficient maximum flow algorithms).

Concretely, we can start with an empty set of middlebox locations $M = \emptyset$, and in each step, we add a middlebox at location m to M, which maximizes the number of totally covered (by middleboxes at locations $M \cup \{m\}$) communicating pairs, without violating capacity and route stretch constraints. The algorithm terminates, when all pairs were successfully assigned to middlebox instances in M. More precisely, for any set of middlebox instances $M \subseteq U$, we define $\phi(M)$ to be the maximum number of communication pairs that can be assigned to M, given the capacity constraints at the nodes and the route stretch constraints. We show that ϕ is non-decreasing and submodular, and $\phi(M)$ – and the corresponding assignment of pairs to middlebox instances in M – can be computed in polynomial time. This allows us to use Wolsey's Theorem [41] to prove an approximation factor of $1 + O(\log \phi_{\max})$, where $\phi_{\max} = \max_{m \in U} \phi(\{m\})$. Since in our case, $\phi_{\max} = \min\{\kappa, |P|\}$ and $|P| \leq n^2$, this implies that Algorithm 1 computes an $O(\log(\min\{\kappa, n\}))$-approximation for the minimum number of middlebox instances that can cover all pairs P (i.e., all pairs can be assigned to the deployed middleboxes).

3.1. Maximum Assignment

In order to compute function $\phi(M)$, for any $M \subseteq U$, we construct a bipartite graph $B(M) = (M \cup P, E)$, where P is the set of communicating pairs. We will simply refer to the middlebox instances $m \in M$ and pairs $p \in P$ in the bipartite graph as the *nodes*. The edge set E connects middlebox instances $m \in M$ to those communicating pairs $p \in P$ which can be routed via m without exceeding the stretch constraint, i.e. $E = \{(m, p) : m \in M, p = (s, t) \in P, d(s, m) + d(m, t) \leq \rho \cdot d(s, t)\}$, where $d(u, v)$ denotes the length of shortest path between nodes u and v in the network. For each p, the set of such middlebox nodes can be computed in a pre-processing step by performing an all-pair shortest paths algorithm to calculate $d(u, v)$ for each u, v in the network, and for each $p = (s, t) \in P$, selecting the nodes $m \in M$ with $d(s, m) + d(m, t) \leq \rho \cdot d(s, t)$.

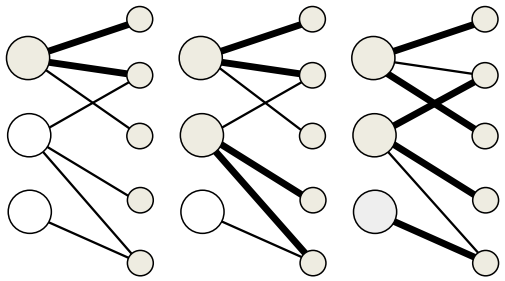

Figure 3: Bipartite graph with possible middlebox locations U (empty: *white*, deployed: *grey*) on the left side and pairs P on the right side of the bipartite graph. The capacity of the middleboxes $\kappa = 2$. Middleboxes are deployed greedily, one-by-one, without requiring relocation of previously mapped middleboxes. Assignment edges are indicated in *bold*. The deployment of the first middlebox *left* creates two assigment edges incident to the middlebox. The deployment of the second middlebox *middle* involves two new assigment edges incident to the second middlebox. Lastly, the botttom middlebox is used to serve the bottom pair, which was previously served by the middle middlebox.

A *partial assignment* $A(M) \subseteq B(M)$ of pairs $p \in P$ to middlebox instances in M is a subgraph of $B(M)$, in which each $p \in P$ is connected to at most one middlebox $m \in M$ by an edge, i.e., $\deg_{A(M)}(p) \leq 1$ where deg denotes the degree. A pair $p \in P$ with $\deg_{A(M)}(p) = 1$ is called an *assigned pair* and with $\deg_{A(M)}(p) = 0$ an *unassigned pair* of a *free pair*. A partial assignment $A(M)$ without free pairs is called an *assignment*. The *size* $|A(M)|$ of a (partial) assignment $A(M)$ is defined as the number of edges in $A(M)$.

Our goal is to compute a partial assignment $A(M)$ of pairs $p \in P$ to middlebox instances in M maximizing the number of assigned pairs. Accordingly, we distinguish between *assignment edges* E_A and *non-assignment edges* $E_{\overline{A}}$, where $E_A \cup E_{\overline{A}} = E$ is a partition of the edge set E of $B(M)$.

Our algorithm ensures that at any moment of time, the partial assignments are *feasible*, i.e., the assignment fulfills the following capacity constraints. The *current load* of a middlebox m in M, denoted by $\lambda(m)$, is the number of communicating pairs served by m according to the current partial assignment $A(M)$. Moreover, we define the *free capacity* $\kappa^*(m)$ of m to be $\kappa^*(m) = \kappa - \lambda(m)$. A (partial) assignment $A(M)$ is feasible if and only if it does not violate capacities, i.e., $\lambda(v) \leq \kappa$, for all v in any middlebox in M.

In order to compute the integer function $\phi(M)$, which is essentially the cardinality of a maximum feasible partial assignment $A^*(M)$, we make use of augmenting paths. Let $A(M)$ be a feasible partial assignment. An *augment-*

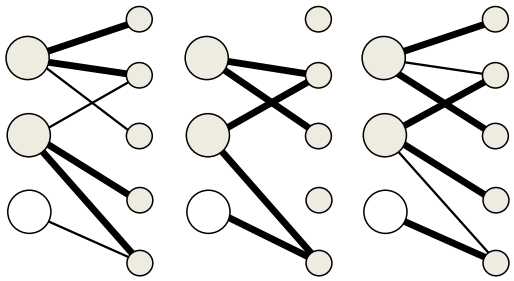

Figure 4: Illustration of augmenting path computation. Bipartite graph with possible middlebox locations U (empty: *white*, deployed: *grey*) on the left side and pairs P on the right side of the bipartite graph. The capacity of the middleboxes $\kappa = 2$. Assignment edges are indicated in *bold*. *Left:* The assignment before deploying the third middlebox. *Middle:* Augmenting path starting at the new middlebox. *Right*: The resulting new assignment.

ing path $\pi = (v_1, v_2, \ldots, v_j)$ relative to $A(M)$ in $B(M)$ starts at a middlebox $m \in M$ with free capacity, ends at a free pair $p \in P$, and alternates between assignment edges and non-assignment edges, i.e.,

1. $v_1 \in M$ with $\kappa^*(v_1) > 0$ and $v_j \in P$ with $\deg_{A(M)}(v_j) = 0$,
2. $(v_i, v_{i+1}) \in B(M) \setminus A(M)$, for any odd i,
3. $(v_i, v_{i+1}) \in A(M)$, for any even i.

An augmenting path relative to $A(M)$ is a witness for a better partial assignment: The symmetric difference $A'(M) = (A(M) \setminus \pi) \cup (\pi \setminus A(M))$ is also a (partial) assignment with size $|A'(M)| = |A(M)| + 1$. Due to the properties of the augmenting path, by this reassignment, one additional pair will be covered by the same set of middlebox instances, without violating node capacities: in $A(M)$, the first node had free capacity and the last node represents a free pair. Furthermore, the degree at each internal node of π remains unchanged, since one incident assignment edge gets unassigned and one incident unassigned edge gets assigned. Therefore, the load of the internal nodes in π remains unchanged. Conversely, suppose that a partial assignment $A(M)$ is not a maximum partial assignment. Let $A^*(M)$ be a maximum partial assignment. Consider the bipartite graph $X = (A(M) \cup A^*(M)) \setminus (A(M) \cap A^*(M))$.

Figure 3 illustrates the greedy placement strategy and Figure 4 shows the augmenting path computation.

Note that an augmenting path must always exist for suboptimal covers. To see this, consider the following reduction to matching: replace each mid-

dlebox node with κ many clones of capacity 1, and assign each pair $p \in P$ to the clones in the canonical way. Feasible assignments constitute a matching in this graph, with node degrees one. Given any suboptimal cover, we find a witness for a augmenting path as follows. We take the symmetric difference of the suboptimal and the optimal solution, which gives us a set of paths and cycles of even length. Thus, a path must exist where the optimal solution has one additional edge.

Augmenting paths can be computed efficiently, by simply performing breadth-first searches in $B(M)$, by using the nodes $m \in M$ with free capacities as the starting nodes.

3.2. Submodularity

The set function $\phi : 2^U \to \mathbb{N}$ is called *non-decreasing* iff $\phi(U_1) \leq \phi(U_2)$ for all $U_1 \subseteq U_2 \in 2^U$, and *submodular* iff $\phi(U_1)+\phi(U_2) \geq \phi(U_1 \cap U_2)+\phi(U_1 \cup U_2)$ for all $U_1, U_2 \in 2^U$. Equivalently, submodularity can be defined as follows (See, e.g. in [38] pp. 766): for any $U_1, U_2 \subseteq U$ with $U_1 \subseteq U_2$ and every $u \in U \setminus U_2$, we have that $\phi(U_1 \cup \{u\}) - \phi(U_1) \geq \phi(U_2 \cup \{u\}) - \phi(U_2)$. This is in turn equivalent to: for every $U_1 \subseteq U$ and $u_1, u_2 \in U \setminus U_1$ we have that $\phi(U_1 \cup \{u_1\}) + \phi(U_1 \cup \{u_2\}) \geq \phi(U_1 \cup \{u_1, u_2\}) + \phi(U_1)$.

Let $U_1 \subseteq U_2$ be two arbitrary subsets of U. Consider a maximum assignment $A(U_1)$ for U_1. Let $A(U_2)$ be a maximum assignment for U_2 obtained from $A(U_1)$ by adding the members $u_2 \in U_2 \setminus U_1$ to U_1 one-by-one, and performing the augmenting path method until we have a maximum assignment for the incremented set $U_1 \cup \{u_2\}$. Let $A^{\Pi}(U_1)$ be the projection of $A(U_2)$ to U_1, i.e., for each $u_1 \in U_1$, $p \in P$, the pair p is assigned to u_1 in $A^{\Pi}(U_1)$ if and only if p is assigned to u_1 in $A(U_2)$. First we show that $A^{\Pi}(U_1)$ is a maximum assignment for U_1.

Lemma 1. *Let $A(U_1)$ be a maximum assignment for U_1. Let $A(U_2)$ be a maximum assignment for U_2 obtained from $A(U_1)$ by adding all $u \in U_2 \setminus U_1$ to U_1 one-by-one, and performing the augmenting path method until we have a maximum assignment for $U_1 \cup \{u\}$. Let $A^{\Pi}(U_1)$ be the projection of $A(U_2)$ to U_1. Then $A^{\Pi}(U_1)$ is a maximum assignment for U_1.*

Proof. By adding $u_2 \in U_2 \setminus U_1$ to U_1 and performing the augmenting path method until an augmenting path exists (i.e., until we obtain a maximum assignment for $U_1 \cup \{u_2\}$), it holds that, for each $u_1 \in U_1$, the number of pairs assigned to u_1 does not change: along the augmenting path each internal node

has one incident assignment edge and one non-assignment edge. This also holds after exchanging the assignment edges and the non-assignment edges, i.e., each assignment edge on the augmenting path becomes a non-assignment edge and vice versa. The degrees of the start and end nodes increases by one. Consequently, the degree of each $u_1 \in U_1$ in $A^{\Pi}(U_1)$ is the same as in $A(U_1)$. Since $A(U_1)$ is a maximum assignment and each u_1 has the same degree in $A^{\Pi}(U_1)$, $A^{\Pi}(U_1)$ must be also a maximum assignment. □

Theorem 2. *Let U be the set of all possible middlebox instance locations. Let $\phi : 2^U \to \mathbb{N}$ be the set function, such that for $M \subseteq U$, $\phi(M)$ is the maximum number of pairs in P that can be assigned to M without violating the capacity constraints. Then ϕ is submodular.*

Proof. We show that for all $M_1, M_2 \subseteq U$, $M_1 \subseteq M_2$ and for all $m \in U \setminus M_2$ we have that

$$\phi(M_1 \cup \{m\}) - \phi(M_1) \geq \phi(M_2 \cup \{m\}) - \phi(M_2). \tag{1}$$

This is equivalent to the definition of the submodularity of ϕ.

Consider a maximum assignment $A(M_1)$ for M_1 and a maximum assignment $A(M_2)$ for M_2 obtained from $A(M_1)$ as described in Lemma 1. Now we add m to M_2. Let $A(M_2 \cup \{m\})$ be the maximum assignment for $M_2 \cup \{m\}$ obtained from $A(M_2)$ by adding m to M_2 and performing the augmenting path method until we have a maximum assignment for $M_2 \cup \{m\}$. Let $A^{\Pi}(M_2)$ be the projection of $A(M_2 \cup \{u\})$ to M_2 and let $A^{\Pi}(M_1)$ be the projection of $A(M_2 \cup \{u\})$ to M_1. Let $|A^{\Pi}(M_2)|$ and $|A^{\Pi}(M_1)|$ be the number of assigned pairs of P in the assignments. By Lemma 1, $A^{\Pi}(M_2)$ is a maximum assignment for M_2 and $A^{\Pi}(M_1)$ is a maximum assignment for M_1. Therefore, $\phi(M_2) = |A^{\Pi}(M_2)|$ and $\phi(M_1) = |A^{\Pi}(M_1)|$. Furthermore, $\phi(M_2 \cup \{u\}) = |A(M_2 \cup \{u\})|$.

Consider the pairs $P_m \subseteq P$ assigned to m in the assignment $A(M_2 \cup \{m\})$. Let $A^{\Pi}(m)$ be the projection of $A(M_2 \cup \{m\})$ to m. Then $|A^{\Pi}(m)| = |P_m|$. Since $A^{\Pi}(M_2)$ contains all elements that are assigned to any element of M_2 in $A(M_2 \cup \{m\})$, clearly $|A(M_2 \cup \{m\})| = |A^{\Pi}(M_2)| + |A^{\Pi}(m)|$, and thus

$$\phi(M_2 \cup \{m\}) - \phi(M_2) = |A^{\Pi}(m)|. \tag{2}$$

On the other side, the assignment $A^*(M_1 \cup \{m\})$ which is obtained as the union of $A^{\Pi}(M_1)$ and $A^{\Pi}(m)$ is a valid assignment for $M_1 \cup \{m\}$. Therefore, $|A(M_1 \cup \{m\})| \geq |A^*(M_1 \cup \{m\})| = |A^{\Pi}(M_1)| + |A^{\Pi}(m)|$, and thus

$$\phi(M_1 \cup \{m\}) - \phi(M_1) \geq |A^{\Pi}(m)|. \tag{3}$$

Algorithm 1: Greedy Algorithm

```
1: init M ← ∅, A(M) ← empty assignment
2: while A(M) is not a feasible assignment do
3:    init m* ← ∅, opt ← 0, tmp ← 0
4:    for each m ∈ U \ M
         (* compute all augmenting paths *)
5:       tmp ← φ(M ∪ {m}) − φ(M)
6:       if tmp > opt then
7:          opt ← tmp, m* ← m
8:       end if
9:    end for
10:   M ← M ∪ {m*}, update A(M)
11: end while
```

Using (2) and (3) we obtain

$$\phi(M_1 \cup \{m\}) - \phi(M_1) \geq \phi(M_2 \cup \{m\}) - \phi(M_2)\,, \tag{4}$$

completing our proof. □

3.3. The Algorithm

Essentially, Algorithm 1 starts with an empty set M and cycles through the possible middlebox locations $m \in U \setminus M$, always deploying the middlebox resulting (with the already deployed ones) in the highest function value ϕ.

Given the submodularity and the augmenting path construction, we have derived our main result. Per middlebox, an augmenting paths problem is solved. Using the *Hopcroft-Karp algorithm*, we can compute all (at most κ many) augmenting paths starting at a newly added middlebox in time $O(\min\{\kappa, \sqrt{|V|}\} \cdot |E|)$, where $|V|$ denotes the number of nodes and $|E|$ the number of edges in $B(M)$.

Theorem 3. *Our greedy and incremental middlebox deployment algorithm computes a $O(\log n)$-approximation.*

3.4. Lower Bound and Optimality

Theorem 3 is essentially the best we can hope for:

Theorem 4. *The middlebox deployment problem is NP-hard and cannot be approximated within* $c \log n$*, for some* $c > 0$ *unless* $P = NP$*. Furthermore, it is not approximable within* $(1 - \epsilon) \ln n$*, for any* $\epsilon > 0$*, unless NP* $\subset$ DTIME$(n^{\log \log n})$.

Proof. We present a polynomial time reduction from the Minimum Set Cover (MSC) problem, defined as follows: Given a finite set S of n elements and a collection C of subsets of S. A set cover for S is a subset $C' \subseteq C$ such that every element in S is contained in at least one member of C'. The objective is to minimize the cardinality of the set cover C'.

Consider an instance of the MSC problem: let $S = \{v_1, ..., v_n\}$ be a set of n elements, $C = \{S_i \subseteq S, i = 1, ..., m\}$. We define the instance of the corresponding middlebox deployment problem in a network $G = (V, E)$ with a set of communicating pairs P and stretch $\rho = 1$ as follows. For each element $v \in S$, we introduce two nodes v_s and v_t in V. For each subset $S_i \in C$, we introduce a node v_{S_i} in V as well. The edge set E of the network $G = (V, E)$ is defined by the following rule: there is an edge $(v_s, v_{S_i}) \in E$ and an edge $(v_{S_i}, v_t) \in E$ iff the corresponding element v is contained in S_i. The set of communicating pairs is defined as $P = \{(v_s, v_t) : v \in S\}$ and the set of potential middlebox locations is defined as $U = \{v_{S_i} : S_i \in C\}$. $G = (V, E)$ is a bipartite graph with partitions U and $\{v_s : v \in S\} \cup \{v_t : v \in S\}$. If $v \in S$ is contained in a set $S_i \in C$ then there is a path of length 2 between the corresponding pair (v_s, v_t) in G. This is also the shortest path between v_s and v_t. In the middlebox deployment problem with stretch $\rho = 1$, a set of nodes $M \subseteq U$ of minimum cardinality must be selected such that between each pair $(v_s, v_t) \in P$ there is a route of length of at most 2 and it contains at least one node of M. By the construction of the network, for each pair $(v_s, v_t) \in P$, there is a route v_s, v_{S_i}, v_t of length 2 in G if and only if $v \in S_i$. Let $M \subseteq U$ be a minimum cardinality solution of the middlebox deployment problem. The node set M implies a minimum cardinality solution for the MSC problem and vice versa. This proves the NP-hardness of the problem.

An important property of the above reduction is that it preserves the approximation factor of the MSC problem. Each deployed middlebox in the solution of the resulting middlebox deployment problem corresponds to a selected set of the original MSC problem. Therefore, the optimal solution of an MSC problem instance and the solution of the corresponding middlebox deployment problem instance have the same cardinality and any k-approximate solution for the middlebox deployment problem corresponds to

a k-approximate solution of the original MSC problem. The solution of the MSC problem can be obtained from the corresponding middlebox deployment solution in polynomial time.

The inapproximability results then follow from the combination of the above reduction, which preserves the approximation factor, and the inapproximability results of the MSC problem by Raz and Safra [35] and by Feige [15]. Raz and Safra [35] proved that the MSC problem is not approximable within $c \log n$, for some $c > 0$, unless $P = NP$. Feige [15] showed the inapproximability within $(1 - \epsilon) \ln n$, for any $\epsilon > 0$, unless $NP \subset \text{DTIME}(n^{\log \log n})$. If we had a polynomial time approximation algorithm for the middlebox deployment problem with an approximation factor better than $(1 - \epsilon) \ln n$, for any $\epsilon > 0$, by the above reduction, we would have a better approximation factor for the MSC problem, as well. □

4. Group and Weighted Variant

There are scenarios in which more than two nodes may have to share a network function. For example, consider the problem of placing a multiplexer for a group of users involved in a teleconference. Or imagine the problem of mapping a shared object or entire virtual server server of a multi-user game: in order to avoid state synchronization overheads, the users should be served from a single location which is also located close to all the users. Also in the context of distributed cloud computing, the problem of placing functionality for larger groups may be relevant. When considering communication groups of heterogeneous size, it is reasonable to assume that the load induced on the network function is also heterogeneous. Furthermore, in reality, different communication pairs will often communicate at different rates, which likely also results in heterogeneous middlebox loads.

In this section, we show that the weighted communication request variant, and therefore also the group variant, can be solved by exploiting an interesting connection to energy-efficient scheduling [16, 17, 23]. As we will see, in order to make this generalization, we will however need to sacrifice the incremental property. Moreover, we need a constant factor resource augmentation (concretely, a factor of 2).

4.1. Formal Model

Instead of considering communication pairs with unit demands we now slightly extend the formal model presented in Section 2.1. Concretely, we gen-

eralize the notion of communication pairs to requests which can be weighted, i.e. different requests may induce different loads on middlebox instances. Hence, a communication request $p \in P$ may now actually be an tuple of arbitrarily many communication partners. Additionally, to capture the stretch constraints, we assume that for each request $p \in P$ a subset of allowed middlebox locations $U_p \subseteq U$ is computed in a preprocessing step. The task is again to determine a minimum set of locations $M \subseteq U$ together with an assignment $\mu : P \to M$, such that:

1. Each request $p \in P$ is connected to a valid middlebox instance, i.e. $\mu(p) \in U_p$ holds.
2. For all middlebox instances $m \in M$ the load does not exceed the capacity κ, i.e. $\sum_{p \in P : \mu(p) = m} d_p \leq \kappa$ holds.

We note that this model naturally generalizes the previously studied model: the problem instance considering communication pairs $P \subseteq V \times V$ with feasible middlebox locations $U \subseteq V$ and stretch ρ can be modeled by setting $d_{(s,t)} = 1$ and $U_{(s,t)} = \{u \in U : d(s,u) + d(u,t) \leq \rho \cdot d(s,t)\}$ for $(s,t) \in R$.

4.2. Relationship to Datacenter Scheduling

There exists an interesting relationship of this generalized problem to scheduling jobs in datacenters. Concretely, Khuller et al. consider in [23] the following scheduling problem. Given are a set of m machines and a set of n jobs which are to be executed. The processing time of job j on machine i is given as $p_{i,j}$ and each machine i has an activation cost a_i. The task is to find a set of machines to activate, such that either (1) the makespan is minimized while the activation costs are bounded by a budget, or (2) the activation costs are minimized while the makespan is bounded by a time budget T. The makespan here refers to the maximum cumulative processing time of jobs assigned to a single machine.

For the latter problem variant, i.e. when minimizing the activation costs given a bound T on the makespan, Khuller et al. [23] present a greedy $(2, \ln n + 1)$ approximation algorithm, such that the solution's activation cost is within a factor of $1 + \ln n$, where n denotes the number of jobs, of the optimal activation cost and the makespan of the assignment is bounded by two times the bound T (if a solution exists).

By a reduction of the generalized (weighted) variant of the network function placement problem to the datacenter scheduling problem, we obtain the following:

Theorem 5. *The generalized network function placement problem can be* $(2, 1 + \ln |U|)$*-approximated via the greedy algorithm of Khuller et al. [23] in polynomial time, i.e., the number of deployed middleboxes exceeds the minimum number of middleboxes by a factor of at most* $(1 + \ln |U|)$ *and the maximum load on each middlebox is at most* $2 \cdot \kappa$*.*

Reduction. Given an instance to the generalized network function placement problem, we model the given instance as a datacenter scheduling instance as follows. The machines correspond to middlebox locations U, and the jobs correspond to the request set R. The processing time $p_{u,p}$ for scheduling job (request) p on machine (middlebox) u is set to equal the demand of the request, i.e. d_p, if p may be assigned to u. In case that request $p \in P$ cannot be assigned to the location u, i.e. if $p \notin U_p$ holds, we set $p_{u,p} = \infty$ such that p can never be scheduled on u. Lastly, the activation costs of all machines are set to 1, i.e. $a_u = 1$ holds for all machines (middlebox locations) $u \in U$, and the bound on the makespan is set to κ. If a solution to the generalized network function placement instance using h middleboxes (not exceeding the middlebox capacity κ) exists, then the greedy approximation of Khuller et al. [23] yields a solution that uses at most $h \cdot (1 + \ln m)$, $m = |U|$, many machines (i.e. middlebox instances) and that exceeds the makespan (capacity) of any machine (middlebox instance) by at most a factor of 2. □

4.3. Greedy Approximation for the Generalized Middlebox Deployment

In the following section we present the greedy approximation algorithm of Khuller et al. [23] in the light of the middlebox deployment problem, i.e. simplified to the case in which activation costs of machines are uniform and job processing times are independent of the machine. We include our adaptation to the middlebox deployment problem for the sake of completeness, since we evaluate its performance in Section 5, as well as to highlight, that the incremental deployment property is lost in this case.

The greedy approximation for the generalized setting works similarly to greedy algorithm presented in Section 3 by iteratively selecting the location maximizing the number of connected requests. However, while for the simpler problem variant augmenting paths could be employed to *exactly* determine the next optimal location, this is not possible in the weighted case: determining the optimal assignment of requests under non-uniform demands is NP-hard (cf. [23, 40]). Instead, given a set $M \subset U$ of middlebox locations, the maximum fractional assignment is computed using the Linear Program 1:

LP 1: Maximum Fractional Assignment

$$\max \quad f(M) = \sum_{p \in P} \sum_{m \in M \cap U_p} x_{m,p}$$

$$\text{subject to} \quad \sum_{m \in M \cap U_p} x_{m,p} \leq 1 \qquad \forall\ p \in P$$

$$\sum_{p \in P : m \in U_p} d_p \cdot x_{m,p} \leq \kappa \qquad \forall\ m \in M$$

$$0 \leq x_{m,p} \leq 1 \qquad \forall\ p \in P, m \in M \cap U_p$$

Algorithm 2: Generalized Greedy Algorithm

```
1: init M ← ∅
2: while |M| < |U| and f(M) ≤ |P| − 1 do
3:    choose m ∈ U \ M s.t. f(M ∪ {m}) − f(M) is maximized
4:    set M = M ∪ {m}
5: end while
6: if f(M) ≤ |P| − 1 then
7:    return 'no solution exists'
8: else
9:    place middleboxes at locations in M
10:   extract request-middlebox assignment from solution to LP 1
11: end if
```

the variable $x_{m,p}$ indicates for each valid combination of (placed) middlebox $m \in M$ and request p the *fractional* assignment of p to m. The constraints enforce that any request is (fractionally) assigned to *at most* one location and that the (fractional) assignments do not violate the middlebox capacities. Accordingly, starting with no placed middleboxes ($M = \emptyset$), the generalized greedy algorithm (see Algorithm 2) selects in each iteration the middlebox $m \in U \setminus M$ maximizing the fractional assignment value $f(M \cup \{m\}) - f(M)$. The algorithm proceeds until strictly more than $|P| - 1$ many requests could be assigned; stopping prematurely is required (and feasible) due to the fractional nature of the solution and the *rounding* procedure to obtain an integral assignment later on (cf. [23]). In particular, to compute the integral assignments of requests P to the selected middleboxes M, the rounding procedure by Tardos and Shmoys [40] is employed:

1. A bipartite graph $B(\vec{x})$ is constructed from the fractional LP solution $\vec{x}$ of LP 1: one color class consists of the request nodes corresponding to P and the other color class represents (copies) of the middlebox locations of M. In particular, each middlebox location $m \in M$ is represented by $\lceil \sum_{p \in P: m \in M \cap U_p} x_{m,p} \rceil$ many nodes. The weighted edges of $B(\vec{x})$ are also created according to the vector $\vec{x}$, such that the sum of weights at each request node (in $B(\vec{x})$) sums to 1 and such that the cumulative weight at the nodes corresponding to a single middlebox location $m \in M$ equals $\sum_{p \in P} x_{m,p}$ (cf. [40] for further details).
2. An (integral) matching X covering all request nodes of minimum cost is computed in $B(\vec{x})$. Such a matching can be easily computed as a minimum-cost flows of value $\lceil f(M) = \sum_{p \in P, m \in M \cap U_p} x_{m,p} \rceil = |P|$.
3. The request j is assigned to middlebox location i if j is connected to a copy of i in the matching X.

The above procedure always yields a matching of cost $\lceil f(M) = \sum_{p \in P, m \in M \cap U_p} x_{m,p} \rceil = |P|$, while the capacities are violated by a factor of at most 2 (by cleverly creating the edges in $B(\vec{x})$) [40].

The correctness of the algorithm follows from the correctness of [23] and [40] and we lastly note that the algorithm does not allow for incremental deployments: whenever a middlebox m is added to M in Algorithm 2, previously connected (accepted) requests might be disconnected (preempted), since solutions to the fractional matching solution computed using LP 1 are computed independently of previous assignments.

Name	Type	$\|V\|$	$\|E\|$
Quest	Continent	20	62
GtsHungary	Country	30	62
Geant	Continent	40	122
Surfnet	Country	50	136
Forthnet	Country	62	124
Telcove	Country	71	140
Ulaknet	Country	82	164

Name	Type	$\|V\|$	$\|E\|$	$\|D\|$
nobel-eu	Continent	28	41	378
cost266	Continent	37	55	1,332
germany50	Country	50	88	662
ta2	Country	65	108	1,869

Figure 5: Topology Zoo Instances (left) as well as SNDlib instances (right) used in the evaluation. $|D|$ denotes the number of defined end-to-end communication requests.

5. Evaluation

In order to complement our formal analysis, we conduct a simulation study comparing the approximation algorithms presented above to the respective optimal solutions computed using Integer Programming (IP). We evaluate the performance of the greedy approximation algorithm in terms of number of installed middleboxes and runtime. Furthermore, we explore the performance of the approximation algorithm for the weighted variant and when incrementally deploying single middleboxes.

Our computational study encompasses more than 10,000 instances, which are all generated at random based on real-world wide-area network topologies. The results of our study indicate that the approximation algorithms presented in this paper typically yield good results and place only marginally more middleboxes than in the optimal solution. However, based on the uniformity of the generation procedure (cf. Section 5.1), we note that our study captures *only* the average case behavior under uniform communication demands between nodes in the network. To enable the future study of our algorithms in different settings, we have made our evaluation framework (Python 3.6) publicly available on GitHub [37].

All experiments were conducted on servers with two Intel XEON L5420 processors (8 cores overall) equipped with 16 GB RAM. The runtimes we report on are wall-clock time.

5.1. Datasets

We have generated two datasets, one for the unweighted approximation algorithm presented in Section 3 and one for the weighted approximation algorithm presented in Section 4.

For the unweighted variant, we use real-world wide-area topologies obtained from the topology zoo collection [24]. As shown in Figure 5 (top), the topologies are either country-wide or continent-wide ISP, backbone, or – in case of Geant – research networks. The topologies were selected so that the number of nodes is equally spread across the range 20 to 82. Additionally, the topologies provide geographical information for nodes, such that communication latencies can be estimated based on the geographical distance.

For each graph, we generate instances as follows: Each of the $|V| \cdot (|V| - 1)/2$ potential communication pairs is selected with probability p, set to 0.2, 0.3, or 0.4, respectively. Hence, the number of expected communication pairs to be created overall is $p \cdot |V| \cdot (|V| - 1)/2$.

To ensure comparability across topologies, all nodes are allowed to host middlebox functionality. The capacity κ is set to $\lceil 2 \cdot (|V| - 1) \cdot p \rceil$: Hence, (in expectancy) around $|V|/4$ many middleboxes are likely to be necessary to serve the communication pairs. All communication pairs have the same (latency) stretch. Concretely we set the maximal allowed latency to $\{1.00, 1.05, 1.10, \ldots 2.50\}$ times the latency of the shortest path between the nodes of the communication pair. For each topology and each probability, we generated 11 scenarios uniformly at random, yielding more than 7,000 instances overall.

We employ a similar approach for the evaluation of the approximation algorithm for the weighted variant. Concretely, we consider four instances (see Figure 5 (bottom)) of the SNDlib [32], which already define communication pairs exhibiting a diverse demand structure. Analogously, we again consider stretches of $\{1.0, 1.05, \ldots, 2.5\}$. To generate multiple instances from a single SNDlib instance, we sample requests by selecting each original communication pair independently at random with a probability of 0.5. We again allow to place middleboxes on all nodes in the network and set the capacity of each of the potential middlebox locations to $4 \cdot D/|V|$, where D denotes the cumulative demand of the chosen requests, such that at least $1/4$ many nodes must be equipped with middleboxes. We generate 3,100 scenarios in total by generating 25 instances per stretch parameter and per topology.

5.2. Baseline Algorithms

Besides employing the approximation algorithms presented above, we use Integer Programs to compute optimal solutions. Integer Program 2 computes optimal solutions for the middlebox deployment problem with unitary demands[1] and works as follows:

For all potential communication pairs $p \in P$ we define $U_p = \{u \in U : (s, u) + d(u, t) \leq \rho \cdot d(s, t)\}$ to denote the feasible set of middlebox locations which may serve p (cf. Section 4.1). Clearly, U_p can be precomputed efficiently. For all potential middlebox locations $u \in U$, we introduce the binary variable $x_u \in \{0, 1\}$, which indicates whether on u a middlebox is placed. For all $p \in P, u \in U_p$, we introduce the binary variable $x_{u,p} \in \{0, 1\}$. The variable $x_{u,p}$ indicates that the pair $p = (s, t) \in P$ is assigned to the node $u \in U_p$, s.t. the path stretch from s to t through u is at most ρ.

[1]The existence of a 0-1 integer linear program, together with our NP-hardness result, also proves the NP-completeness [22].

Integer Program 2: Minimizing the Number of Middleboxes

$$\min \sum_{u \in U} x_u \tag{5}$$

$$\text{subject to} \quad \sum_{u \in U_p} x_{u,p} = 1 \qquad \forall\, p \in P \tag{6}$$

$$\sum_{p \in P : u \in U_p} x_{u,p} \leq \kappa \cdot x_u \;\forall\; u \in U \tag{7}$$

$$x_u \in \{0,1\} \;\; \forall u \in U\,, \qquad x_{u,p} \in \{0,1\} \;\; \forall\, p \in P, u \in U_p \tag{8}$$

Integer Program 3: Maximum Assignment for exactly n Middleboxes

$$\max \sum_{p \in P, u \in U_p} x_{u,p} \tag{9}$$

$$\text{subject to} \quad \sum_{u \in U_p} x_{u,p} \leq 1 \qquad \forall\, p \in P \tag{10}$$

$$\sum_{p \in P : u \in U_p} x_{u,p} \leq \kappa \cdot x_u \qquad \forall\, u \in U \tag{11}$$

$$\sum_{u \in U} x_u = n \tag{12}$$

$$x_u \in \{0,1\} \;\; \forall u \in U\,, \qquad x_{u,p} \in \{0,1\} \;\; \forall\, p \in P, u \in U_p \tag{13}$$

The Objective Function (5) requires that a minimum cardinality middlebox set must be selected. Constraints (6) declare that each pair $p = (s,t) \in P$ is assigned to exactly one node $u \in U_p$, hence satisfying the constraint on the stretch. Constraints (7) describe that the capacity limit κ must not be exceeded at any node, and nodes $u \in U$ which are not selected in the solution M (where the corresponding variable x_u becomes 0 in the solution) are not used by any pair $p \in P$.

This Integer Program can easily be adapted for the weighted (and or group) variant by adapting Constraint (7). Concretely, Constraint 7 needs to include the additional factor d_p, denoting the demand of request p:

$$\sum_{p \in P} d_p \cdot x_{up} \leq \kappa \cdot x_u \qquad \forall u \in U\,.$$

As we are also interested in studying the opportunities arising in incremental

deployment scenarios, we also introduce IP 3 which computes the maximum assignment for any given number of middleboxes. Concretely, given a number $n \in \mathbb{N}$ of middleboxes to activate (see Constraint 12), the number of connected communication pairs is maximized (see Constraint 9), while ensuring that a communication pair may at most be assigned to exactly one middlebox (see Constraint 10).

5.3. Runtime and Number of Middleboxes

We first study the runtime and the empirical approximation factor of the approximation algorithm 1 on the randomly generated topology zoo instances. Our implementation (cf. [37]) may use multi-threading for the computation of the middlebox-selection in Algorithm 1 (Lines 4-9). The computation of the Integer Program solutions is implemented via Gurobi 6.5 using a single thread.

In Figure 6 (center), the average runtime of the greedy (G) approximation algorithm with 1 and 8 threads, and of the Integer Program (IP) is depicted. Here, each data point represents the aggregate of $31 \times 11 = 341$ experiments. The average runtime of the sequential greedy-algorithm lies below the one of the IP for 20 and 30 nodes, and the greedy algorithm with 8 threads clearly outperforms the IP on the topologies with 62, 71 and 82 nodes. On the largest topology the computation of the greedy algorithm can be sped up by a factor of around 5, by using 8 threads. Furthermore, the runtime of the IP is on the largest topology one magnitude higher than the one of the 8-threaded greedy algorithm. The right plot of Figure 6 depicts the runtime of the 8-threaded greedy variant and the IP on the largest topology

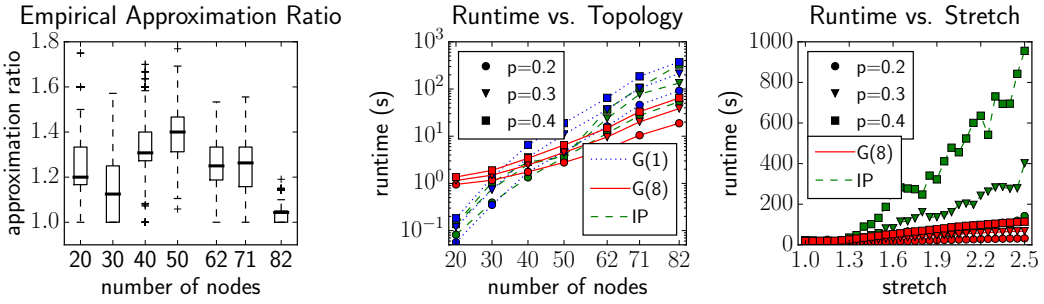

Figure 6: *Left:* Empirical approximation ratio, i.e. the number of placed middleboxes by the greedy approximation divided by the optimum number. *Center and left:* Runtime of the algorithms as a function of the different topologies and as a function of the stretch on the Ulaknet topology.

as a function of the stretch. Starting at a stretch of 1.3, the runtime of the IP increases dramatically. This is due to the fact that by increasing the stretch, the number of potential middleboxes, serving a communication pair, increases. In fact, on the largest topology, the IP consisted of up to 90k variables.

The right box plot of Figure 6 depicts the quality of the solutions found by the approximation algorithm, namely the number of middleboxes opened divided by the optimal number of middleboxes computed by the IP. The median always lies below 1.5 and the maximum is close to 1.8.

5.4. Weighted Requests

Next, we study the performance of the approximation algorithm discussed in Section 4, for weighted (or group) problems (cf. Algorithm 2). As discussed in Section 5.1, we use SNDlib instances exhibiting a diverse demand structure (see the left plot of Figure 7).

As a first result, the center plot of Figure 7 shows the averaged relative number of deployed middleboxes of the greedy algorithm with respect to the optimal solution of the Integer Program. The greedy approximation algorithm only seldomly opens – on average – more than 20% middleboxes more than the optimal algorithm. Note that, in some cases, the number of deployed middleboxes lies even beneath the optimal one. This is possible, as the considered algorithm may (cf. Figure 7 *right*) violate the middlebox capacity up to a factor of 2. Indeed, the approximation algorithm violates

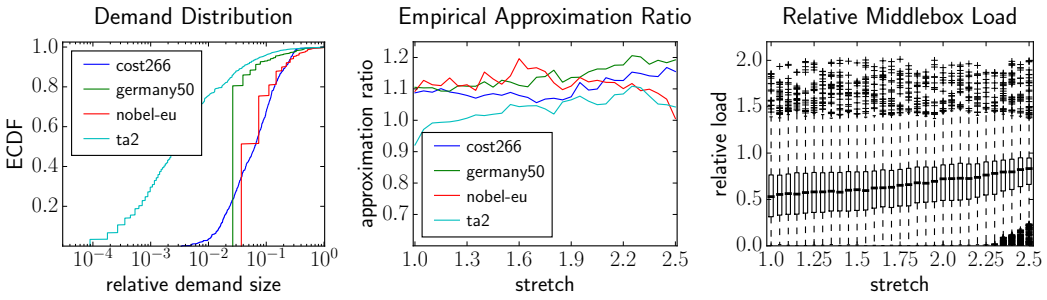

Figure 7: *Left:* The demand structure on the SNDlib instances as empirical cumulative distribution function. *Center:* The relative number of opened middleboxes compared to the solution of the baseline Integer Program. *Right:* The relative load, i.e. load divided by the available capacity, on the middleboxes computed by the $(2, 1 + \ln n)$-approximation algorithm.

capacities in less than 25% of all cases and the median load lies beneath 100%.

5.5. Incremental Middlebox Activation

Lastly, we study scenarios in which middleboxes are added one after another. Concretely, we assume that all communication pairs are known in advance while the network operator can only incrementally install single middleboxes (e.g. due to the associated cost). The greedy algorithm always places one additional middlebox while not being allowed to change previously selected middlebox locations. We study the number of assigned communication pairs of the greedy algorithm compared to the optimum number of assignments when middlebox locations may be arbitrarily selected. To compute the optimum number of assignments the Integer Program 3 is used.

For this set of experiments, we again use the unit-demand instances based on the topology zoo networks (cf. Section 5.1), while only fixing the probability to create communication pairs to $p = 0.3$.

We present the results of this set of experiments in Figure 8. The left plot depicts the relative difference of assigned communication pairs when using the greedy algorithm compared to the Integer Programming (IP) baseline. Concretely, the relative difference is defined as $(\phi_{\text{IP}} - \phi_{\text{G}})/\phi_{\text{IP}}$, where ϕ_{IP} and ϕ_{G} denotes the number of assigned communication pairs of the IP and the greedy algorithm, respectively. The rows of the left plot averages the results for all scenarios having the same number of minimal middleboxes to serve all communication pairs. The number of scenarios averaged in this way is

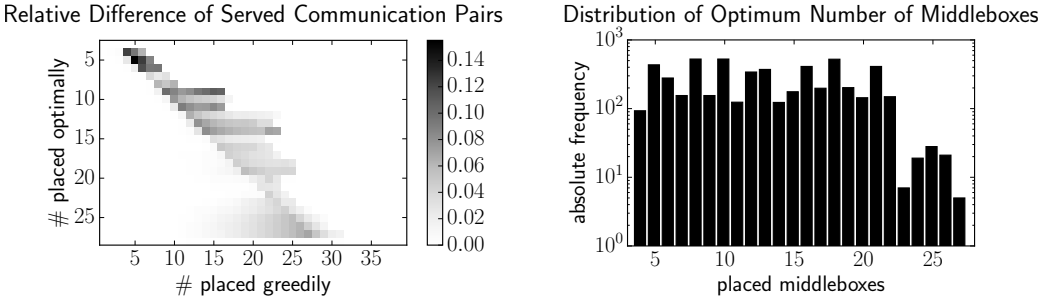

Figure 8: *Left:* The averaged relative difference in the number of served communications pairs depending on the number of optimally placed middleboxes (y-axis). *Right:* Minimum number of middleboxes computed by the Integer Program; the results in the left plot are averaged over the corresponding number of scenarios accordingly.

depicted in the right plot of Figure 8: the number of scenarios for which 10 middleboxes suffice and which are averaged in the row 10 is around 500.

Considering any row, we see that the greedy algorithm assigns nearly always as many communication pairs until the number of middleboxes reaches the optimum (minimal) number of middleboxes. After coming close to the optimum number of middleboxes, the relative difference in assignments reaches a maximum valiue of 0.15 and then diminishes only slightly with each additional greedily placed middlebox. The ability to (re-)place middleboxes in an arbitrary fashion hence only becomes important when sufficiently many middleboxes were already placed to serve almost all communication pairs.

6. Summary and Conclusion

This paper initiated the study of the network function placement problem which is motivated by the increasing flexibilities of modern virtualized networked systems. Our main contribution is a combinatorial proof of the submodularity of this problem and an incremental log-approximation network function placement algorithm. We also initate the study of a greedy approach for a weighted group-version of the problem. Our simulation results show that this approach computes a nearly optimal placement for real world network instances.

We understand our work as a first step, and believe that our paper opens several interesting directions for future research. In particular, it will be interesting to know whether good approximations exist for the incremental deployment of entire group requests as well.

Acknowledgement. We thank Alexander Elvers for his extensive work on refactoring our code [37] for publication.

[1] K. Aardal, P. L. van den Berg, D. Gijswijt, and S. Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242(2):358–368, 2015.

[2] A. Abujoda and P. Papadimitriou. MIDAS: middlebox discovery and selection for on-path flow processing. In *Proc. 7th COMSNETS*, pages 1–8, 2015.

[3] M. Bagaa, T. Taleb, and A. Ksentini. Service-aware network function placement for efficient traffic handling in carrier cloud. In *Proc. IEEE*

Wireless Communications and Networking Conference (WCNC), pages 2402–2407, 2014.

[4] N. Bansal, R. Krishnaswamy, and B. Saha. On capacitated set cover problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Proceedings*, pages 38–49, 2011.

[5] F. A. Chudak and D. P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical Programming*, 102(2):207–222, 2005.

[6] J. Chuzhoy and J. Naor. Covering problems with hard capacities. In *Proc. IEEE FOCS*, pages 481–489, 2002.

[7] J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.

[8] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[9] D. Levin et al. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise network. In *Proc. USENIX ATC*, 2014.

[10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[11] G. Even, M. Medina, and B. Patt-Shamir. Online path computation and function placement in sdns. In *Proc. 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2016.

[12] G. Even, M. Rost, and S. Schmid. An approximation algorithm for path computation and function placement in sdns. In *Proc. 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2016.

[13] S. Fayazbakhsh et al. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proc. ACM HotSDN*, 2013.

[14] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. *Queue*, 11(12), 2013.

[15] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[16] L. Fleischer. Data center scheduling, generalized flows, and submodularity. In *Proc. 7th Workshop on Analytic Algorithmics and Combinatorics, (ANALCO)*, pages 56–65, 2010.

[17] M. Gairing, B. Monien, and A. Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380(1-2):87–99, 2007.

[18] A. Gember-Jacobson et al. OpenNF: Enabling innovation in network function control. In *Proc. ACM SIGCOMM*, pages 163–174, 2014.

[19] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[20] R. Hartert, P. Schaus, S. Vissicchio, and O. Bonaventure. Solving segment routing problems with hybrid constraint programming techniques. In *Proc. International Conference on Principles and Practice of Constraint Programming (CP)*, 2015.

[21] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[22] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. 1972.

[23] S. Khuller, J. Li, and B. Saha. Energy efficient scheduling via partial shutdown. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1360–1372, 2010.

[24] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 –1775, october 2011.

[25] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. In *Proc. USENIX Annual Technical Conference (ATC)*, pages 333–345, 2014.

[26] T. Lukovszki, M. Rost, and S. Schmid. It's a match! near-optimal and incremental middlebox deployment. *ACM SIGCOMM Computer Communication Review (CCR)*, 46(1):30–36, 2016.

[27] T. Lukovszki and S. Schmid. Online admission control and embedding of service chains. In *Proc. 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 104–118, 2015.

[28] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[29] M. Markovitch and S. Schmid. Shear: A highly available and flexible network architecture: Marrying distributed and logically centralized control planes. In *Proc. 23rd IEEE International Conference on Network Protocols (ICNP)*, 2015.

[30] S. Mehraghdam, M. Keller, and H. Karl. Specifying and placing chains of virtual network functions. In *Proc. IEEE CloudNet*, pages 7–13, 2014.

[31] J. B. Orlin. Max flows in $O(nm)$ time, or better. In *Proc. ACM STOC*, pages 765–774, 2013.

[32] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. Sndlib 1.0 – Survivable Network Design Library. *Networks*, 55(3):276–286, 2010.

[33] Z. Qazi et al. SIMPLE-fying middlebox policy enforcement using SDN. In *Proc. ACM SIGCOMM*, pages 27–38, 2013.

[34] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo. Towards making network function virtualization a cloud computing service. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 89–97, 2015.

[35] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In *Proc. ACM STOC*, pages 475–484, 1997.

[36] S. Raza, G. Huang, C.-N. Chuah, S. Seetharaman, and J. P. Singh. Measurouting: A framework for routing assisted traffic monitoring. *IEEE/ACM Trans. Netw.*, 20(1):45–56, 2012.

[37] M. Rost and A. Elvers. . https://github.com/submodular-middlebox-depoyment/submodular-middlebox-deployment.

[38] A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer-Verlag, 2003.

[39] J. Sherry et al. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proc. ACM SIGCOMM*, pages 13–24, 2012.

[40] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(3):461–474, Dec. 1993.

[41] L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.