The Wide-Area Virtual Service Migration Problem: A Competitive Analysis Approach

Marcin Bienkowski[1], Anja Feldmann[2], Johannes Grassler[2],
Gregor Schaffrath[2], Stefan Schmid[2]

[1] Institute of Computer Science, University of Wrocław, Poland
mbi@cs.uni.wroc.pl
[2] Telekom Innovation Laboratories (T-Labs) & TU Berlin, D-10587 Berlin, Germany
{anja,jgrassler,grsch,stefan}@net.t-labs.tu-berlin.de

***Abstract*—Today's trend towards network virtualization and software-defined networking enables flexible new distributed systems where resources can be dynamically allocated and migrated to locations where they are most useful. This article proposes a competitive analysis approach to design and reason about online algorithms that find a good tradeoff between the benefits and costs of a migratable service. A competitive online algorithm provides worst-case performance guarantees under *any* demand dynamics, and without any information or statistical assumptions on the demand in the future. This is attractive especially in scenarios where the demand is hard to predict and can be subject to unexpected events.**

As a case study, we describe a service (e.g., an SAP server or a gaming application) that uses network virtualization to improve the Quality-of-Service (QoS) experienced by thin client applications running on mobile devices. By decoupling the service from the underlying resource infrastructure, it can be migrated closer to the current client locations while taking into account migration costs. We identify the major cost factors in such a system, and formalize the *wide-area service migration problem*. Our main contribution are a randomized and a deterministic online algorithm that achieve a *competitive ratio* of $O(\log n)$ in a simplified scenario, where n is the size of the substrate network. This is almost optimal. We complement our worst-case analysis with simulations in different specific scenarios, and also sketch a migration demonstrator.

I. Introduction

Virtualization is one of the main innovation motors in the Internet. Essentially all cloud resources and datacenters are highly virtualized today, and an unprecedented amount of resources (typically in the form of *Virtual Machines* (VMs)) can be allocated on demand and for a limited time period only. Naturally, in the network core, the virtualization trend is slower, but also here we witness an evolution towards more "virtualized" or "programmable" forms of networking, and software-defined networking (SDN) and OpenFlow initiatives attract attention also from the ISP side. Recently, Google surprised the world with its announcement to have switched their backbone to SDN.

We believe that within the next ten years, many ISPs will shift towards more flexible forms of networking. For instance, we expect ISPs to use network virtualization technologies to improve the resource allocation or energy consumption in their network, and to offer new services which are aware of and adaptive to the users' demand. Thus, ISPs can monetize their infrastructure more effectively.

The flexibility introduced by network virtualization technology raises interesting research challenges. For example, the possibility to seamlessly move services closer to the users can be exploited to improve Quality-of-Service/Quality-of-Experience (QoS/QoE) parameters. However, as migration comes at a certain cost, good strategies are required to decide on when and where to move services. This article revolves around this question and pursues an *online algorithm and competitive analysis approach* which is attractive especially when the user demand is hard to predict.

Concretely, we attend to a scenario where an ISP uses network virtualization technology to offer an SAP or a *game server* service to mobile thin clients. We assume that the service request patterns change over time, e.g., due to commuting users, due to timezone effects, or due to unexpected events in sports or politics, rendering it worthwhile to migrate the service to different locations. For instance, it can make sense to transfer a service from China to Europe at night, to improve the access to the service both in terms of latency as well as cost (e.g., due to roaming) [18].

While moving services close to clients can improve access latencies, migration comes at a cost. For example, the bulk-data transfer imposes load on the network and may cause a service disruption. One of the main parameters influencing this cost is the available bandwidth along the migration path in the substrate network [10]. If virtual networks (*VNets*) are provisioned across administrative domains belonging to multiple infrastructure providers, (inter provider) migration can also entail certain transit (or *roaming*) costs.

Intuitively, the benefits from virtualization are higher the lower the migration cost (relative to the latency penalty). Also, a predictable access pattern may ease migration. However, if user arrival patterns can be chaotic, uncertainty about future arrivals has to be explicitly incorporated in the model. This article seeks to gain insights into these tradeoffs.

The *competitive analysis framework* is the classic formal tool to study algorithms that deal with a dynamic input (i.e., a dynamic demand) that is revealed in an online fashion and cannot be predicted. In competitive analysis, the performance or cost of a so-called *online algorithm* is compared to an optimal *offline algorithm* that has complete knowledge of the input *in advance*: the ratio of the two costs *in the worst-case* is called the *competitive ratio*. The competitive ratio is a conservative measure that does not rely on any statistical assumptions or prediction models.

A. Our Contribution

This article proposes to study online service migration from a competitive analysis perspective and provides a general formalization of this problem. We initiate the rigorous analysis of a simplified model where the migration costs mainly depend on the available bandwidth along the migration path rather than, e.g., the (hop) length. The available bandwidth determines the *service interruption time*. Such interruptions are still common especially in a wide-area context with limited shared infrastructure.

We make the following technical contributions:

1) We introduce the service migration problem and formalize it in the competitive analysis framework.
2) For a single server setting, we present a randomized online algorithm MIX and a deterministic online algorithm CEN that achieve a *competitive ratio* of $O(\log n)$ in the worst case, i.e., without any knowledge of the future demand, where n is the substrate size. While MIX is a reincarnation of the randomized configuration change algorithm known from Metrical Task Systems [19], CEN exploits the graph properties by migrating the service to a center of gravity.
3) We show that the competitive ratio of MIX and CEN are almost optimal in the sense that we can prove that there does not exist any online algorithm whose competitive ratio is smaller than $\Omega(\log n / \log \log n)$. As we will discuss, the $\Omega(\log n)$ lower bound from the related *n-point uniform space metrical task system* problem does not apply here, and hence the question whether online algorithms with a competitive ratio $o(\log n)$ exist remains open.
4) We describe how a standard dynamic program can be used to compute very general optimal offline solutions. For example, these solutions can be used in our simulations to empirically evaluate the "competitive ratios" (in the context of a specific scenario, we will misuse the term competitive ratio to simply denote the *cost ratio* between the online and offline algorithm).
5) To complement the formal analysis, we report on our simulation results in different settings. These results indicate that our algorithms adapt well to moderate request dynamics and correlation. The competitive ratio may grow slower in the network size on average than predicted by our formal analysis. For comparison with the purely online algorithms, we also introduce the TIMM algorithm that exploits the specific simulation scenario, and discuss its benefits and limitations.
6) To show that our approach is relatively general and that our techniques can be extended, we initiate the discussion of more complex migration models capturing multi-provider scenarios and scenarios where a service is realized with redundant servers.
7) Finally, we describe the "proof-of-concept" migration demonstrator we implemented in our own network virtualization prototype. It shows how interactively added users can change the center of gravity of demand and trigger migration of the latency-sensitive streaming service. Due to the scarce resources, the migration of the streaming service entails additional migrations: already embedded virtual networks need to be moved to other machines to free up resources for the service.

B. Article Organization

This article is organized as follows. After a literature review in Section II, Section III first introduces the envisioned system architecture in our network virtualization prototype; subsequently, the formal model is presented in detail. We describe and analyze our competitive online algorithms in Section IV. Moreover, using the general optimal offline algorithm of Section V as a yardstick to evaluate performance, we complement our formal insights by reporting on our simulation results Section VI. We discuss extensions for our algorithms in Section VII. Section VIII gives an overview of our demonstrator. We conclude our contribution in Section IX.

II. Related Work

Network Virtualization. There has been a significant interest in virtual networks over the last years, which is manifested in the various European (e.g., 4WARD or Trilogy), American (e.g., Clean Slate or GENI) and Asian (e.g., AKARI) projects. Network virtualization enables the co-existence of innovation and reliability [29]. Virtual networks can be realized on different layers and can be supported by new SDN technologies such *OpenFlow* [23]. For a more detailed survey on the subject, please refer to [13].

Embedding. A major challenge in network virtualization is the *embedding* [24] of VNets, that is, the question of how to efficiently and on-demand assign incoming service requests onto the topology. Due to its relevance, the embedding problem has been intensively studied in various settings, e.g., for an offline version of the embedding problem see [22], [28], for an embedding with only bandwidth constraints see [16], for heuristic approaches without admission control see [31], or for a simulated annealing approach see [27]. Lischka and Karl [21] present an embedding heuristic that uses backtracking and aims at embedding nodes and links concurrently for improved resource utilization. Such a concurrent mapping approach is also proposed in [12] with the help of a mixed integer program. Finally, several challenges of embeddings in wireless networks have been identified by Park and Kim [25]. The survey by Belbekkouche [4] provides a nice overview of allocation and embedding algorithms.

Migration and Online Algorithms. In this article, we study the question of how to dynamically embed or migrate virtual servers [26] in order to efficiently satisfy connection requests arriving online at any of the network entry points, and thus use virtualization technology to improve the quality of service for mobile nodes. The relevance of this subproblem of the general embedding problem is underlined by Hao et al. [18] who show that under certain circumstances, migration of a Samba front-end server closer to the clients can be beneficial even for bulk-data applications. The possibility to migrate services without service interruption and even live has interesting implications and poses new challenges. For instance, see [11] for a recent attempt to model tradeoffs (in the cloud).

To the best of our knowledge, [6] and [2] (for online server migration), and [3], [15] (for online virtual network embeddings) are the only works to study network virtualization problems from an online algorithm perspective. The formal competitive migration problem is related to several classic optimization problems such as facility location, k-server problems, or online page migration. All these problems are a special case of the general *Metrical Task System* (e.g., [8], [9]) for which there is, e.g., an asymptotically optimal deterministic $\Theta(n)$-competitive algorithm, where n is the state (or "configuration") space; or a randomized $O(\log^2 n \cdot \log\log n)$-competitive algorithm given that the state space fulfills the triangle inequality: the randomized algorithm proceeds via a (well separated) tree (or *ultrametric*) approximation for the general metric space (in a preprocessing step) and subsequently solves the problem on this distorted space.

In particular, when considering networks with homogeneous resources, our problem can be formulated as an *n-point uniform space metrical task system* (where all distances are equal). From this perspective, MIX is a reincarnation of the randomized configuration change algorithm (see, e.g., [19]). However, due to the topological constraints of the access costs, our service migration problem can achieve a better performance. Indeed, the center of gravity approach by CEN exploits this topological structure which gives a deterministically logarithmic competitive ratio.

The *page migration problem* (e.g., [5]) occurs in managing a globally addressed shared memory in a multiprocessor system. Each physical page of memory is located at a given processor, and memory references to that page by other processors are charged a cost equal to the network distance. At times, the page may migrate between processors, at a cost equal to the distance times a page size factor. The problem is to schedule movements on-line so as to minimize the total cost of memory references. In contrast to these page migration models, we differentiate between access costs that are determined by latency and migration costs that are determined by network bandwidth.

Researchers have also initiated the study of online frameworks that go beyond purely worst-case competitive analysis. For example, prediction models for the future requests or demand have been introduced, for example the so-called *Receding Horizon Control* or the *Model Predictive Control*. [20] We, in this paper, also made a first step towards combining purely online algorithms for our problem with algorithms that exploit request patterns, namely the TIMM strategy. However, a rigorous study of such extensions is left for future research.

Finally, as discussed in Section IV-C, there is a relationship between server migration and *online function tracking* [7], [30], which can be exploited to derive lower bounds.

III. A FLEXIBLE SERVICE PROVIDER

This section defines the scope of our work, and provides an overview of the envisioned service and its operational cost.

A. Economic Roles and Migration

We envision a network virtualization environment where services are offered and realized by different *economic roles*, see [29] for a detailed discussion. In a nutshell, we assume that the physical network, or more generally: the *substrate network*, is owned and managed by one or several *Physical Infrastructure Providers* (PIP) while virtual network abstractions are offered by so-called *Virtual Network Providers* (VNP). VNPs can be regarded as resource *brokers*, buying and combining resources from different PIPs. The virtual network is operated by a so-called *Virtual Network Operator* (VNO). Finally, there is a *Service Provider* (SP) specifying and offering a flexible service.

We attend to a scenario where a Service Provider offers a migratable service which is accessed by (mobile) *terminals*. The terminals connect to the service via *access points* (of fixed locations).

Depending on the specific use case and role, service cost metrics and scopes can differ. For example, for a *PIP*, communication between access points (or terminals for that matter) and the service entails a traffic load on the substrate links: the access cost will depend on the topological distance traveled by the requests. The migration cost will involve reconfiguration costs as well as costs for actually moving the service. In a *VNP* environment, hosting tariffs can be dynamic and depend on the actual resource usage, or, if the VNP role is studied in the context of a sub-structured provider, the PIP's access costs may translate into monetary costs on the VNP side. For VNPs, both service level and resource usage costs can be abstracted by contractual relationships. We deem it likely that a PIP will simplify tariffs for migration by offering package prices derived from its own (e.g., worst case) costs. The VNP may incur additional costs in case of migrations across providers (e.g., depending on the number of transit providers). Regarding the *VNO or SP*, apart from usage-based hosting tariffs the service may incur access costs in terms of latency the VNO has to account for (which may also loosely relate to hosting costs on the provider side). In this article, we will adopt this as a prime motivator for migrations. Migration destinations may be access points (e.g., position requirements), or chosen transparently by providers as a function of link requirements (e.g., latency constraints).

While especially PIP level cost factors can vary, the knowledge and capability to find good approximations for projected costs are fundamental to every business. We will assume that the entity applying our online algorithms is capable of finding reasonable generic approximations of migration costs.

B. Formal Model

This section introduces our formal model. The model is presented relatively generally, while the main algorithmic contribution of this article will be dedicated to a single server and single PIP scenario.

Our formal model considers a substrate network $G = (V, E)$ managed by one or multiple substrate providers (PIPs). Each of the n substrate nodes $v \in V$ has a certain capacity $\omega(v)$ associated with it (e.g., the number of CPU cores, the memory size, or the bus speed). Similarly, each link $e = (u, v) \in E$, with $u, v \in V$, is characterized by a latency $\lambda(e)$ and the available bandwidth for migration, denoted by the capacity $\omega(e)$.

In addition to the substrate network, there is a set T of external machines (the mobile thin clients or simply *terminals*) that access the network G by issuing requests to a virtualized *service* hosted on (up to) k virtual *servers* S. (This article will mainly focus on a single-server scenario, i.e., $k = 1$. Hence we will often use the terms service and server interchangeably.) In order for the terminals T to access the service, a fixed subset of nodes $A \subseteq V$ serve as *Access Points* where machines in T can connect to G.

Due to the mobility and on/off dynamics of the terminals, the request patterns of the access points can change frequently, which may trigger the migration algorithm. We define σ_t to be the multi-set of (arbitrary) requests at time t where each element specifies an access point $a \in A$ from which the request originates. We will assume that requests are always routed to the closest server.

The service access cost Cost_{acc} for a request $r \in \sigma_t$ from access point $a \in A$ to server $s \in S$ (along path $p_r : a \rightsquigarrow s$) at time t is given by the delay $\text{delay}(r) = \sum_{e \in p_r} \lambda(e)$, and the server load $\text{load}(s) = h(\omega(s), \eta(s,t))$ induced by the number of concurrent requests $\eta(s,t)$ served by s at time t, i.e.,

$$\text{Cost}_{\text{acc}}(t) = \sum_{r \in \sigma_t} f\left(\text{delay}(r), \text{load}(s)\right)$$

The migration cost Cost_{mig} of a server s is composed of the opportunistic cost of service outage time, bulk data transfer cost, or roaming costs (e.g., the number of transit providers along the migration path). Thus the cost depends on the available bandwidth $\omega(p_{\text{mig}})$ along the migration path p_{mig}, the size $\text{size}(s)$ of server s, the length ℓ (e.g., in terms of hops) of the path p_{mig}, and the *transit costs* relating to the number x of transit PIPs on the path. Assuming some function g combining these costs, we can write for the migration cost of a server $s \in S$:

$$\text{Cost}_{\text{mig}}(t) = g(\omega(p_{\text{mig}}), \text{size}(s), \ell, x)$$

We, in this paper, particularly emphasize the service outage time: During a long migration period, the users may not experience an optimal service (e.g., can only access stale data), or no service at all. For instance, in our network virtualization prototype implementation [29], a live stream is interrupted during the movement of the streaming server. (See also the discussion in the demonstrator section (Section VIII).)

If $k > 1$ redundant servers are hosted, the number of running servers may also be optimized. Assuming a running cost of Cost_{run}, we can trade off the service quality with the operational costs. Basically, a server may be in three different states: *not in use*, *inactive*, and *active*. If a server is not in use, there are no costs. An inactive server comes at a certain cost C_{in} per time: this cost includes storing the application software (e.g., a game) plus certain maintenance costs. The running cost of an active server C_{ac} is larger, as it also includes CPU costs, state maintenance cost (e.g., RAM state), or bandwidth costs. In order to startup a server which is not in use, a fixed creation cost c will be assumed. For instance, this cost captures the installation of the Linux box and the template (copy if already on disk or download from an NFS share), configuration of the template (e.g., setting up IP addresses manually or via DHCP), starting the server etc. We assume that the cost of changing from inactive to active state is negligible.

C. Dynamic Demand

Our model so far lacks one additional ingredient, the *request dynamics*: How are the requests σ_t distributed over time?

The service access pattern can change for various reasons, including time-zone effects, user mobility, or more unexpected events. Of course, the extent to which a system can benefit from virtual network support and migration depends on the request dynamics: the more correlated the demand, the better it can be served. Given rapid changes it may be best to place the server in the middle of the network and leave it there. On the other hand, if the changes are slower or can be predicted, it can be worthwhile to migrate the server to follow the mobility pattern. This constitutes the trade-off motivating this article.

Note that even if a dynamic access pattern may be due to user mobility, and users travel relatively slowly between access points, these geographical movements may not translate to (and hence be reflected in) the topology of the substrate network: the substrate topology may only loosely correlate with the geography.

Thus, we, in the theoretical part of this article, take a conservative standpoint and will assume *arbitrary* request sets σ_t, where σ_t is completely independent of σ_{t-1}. However, in our simulations, we will consider a "move with the sun" scenario which features topological and temporal dependencies.

D. Objective: The Competitive Ratio

This article proposes an online algorithm and competitive analysis approach to the service migration problem. We will describe online algorithms which at any time t_0, have to take migration decisions without any knowledge of requests σ_t arriving at time $t > t_0$. In order to evaluate the performance of an online algorithm ON, we compare its cost to an optimal offline algorithm OFF, in the worst case. This is known as the *competitive ratio* ρ:

Definition III.1 (Competitive Ratio ρ)**.** *The* competitive ratio *ρ of an online algorithm* ON *is defined as the worst-case cost ratio, overall possible input sequences σ, compared to an optimal offline algorithm* OFF *that knows σ in advance:*

$$\rho = \max_{\sigma} \frac{\mathit{Cost}_{\text{ON}}(\sigma)}{\mathit{Cost}_{\text{OFF}}(\sigma)} \quad .$$

An online algorithm is ρ-competitive if its competitive ratio is at most ρ.

It is standard in the metrical task system literature to assume a *1-lookahead model*: the online algorithm is allowed to make its migrations *after* learning the requests of the current time step, but *before* actually serving them. We will make use of this model also in our article. (Note that under the reasonable assumption that the migration cost is larger than any access cost of a single time step, performing a migration after serving requests does not change the competitive ratio much.)

IV. Competitive Online Algorithms

We will first investigate the service migration problem in the most basic model: we assume that the service is realized with only one server ($k = 1$) and within a single PIP, that the migration cost β is given by the available bandwidth along the migration path which we assume to be constant, and that the access cost is given by the latency along the access path to the service. More formally, we will assume that $\text{Cost}_{\text{mig}}(t) = \beta$ and that $\text{Cost}_{\text{acc}}(t) = \sum_{r \in \sigma_t} d(r)$, where $d(r)$ is the sum of link latencies on the shortest path between the server and the request r.

This section presents two competitive service migration algorithms for this scenario: the randomized algorithm Mix and the deterministic algorithm Cen. We will also formally prove an almost tight lower bound, showing that the algorithms and analyses are almost optimal.

Before presenting our algorithms and formal proofs in detail, let us first provide some intuition. The underlying idea of both Mix and Cen is to strike a balance between access and migration cost. (This is a standard principle in the online algorithm design.) That is, we try to migrate only if the migration cost is "amortized" with respect to access costs. Both Mix and Cen migrate the service to locations which were better (w.r.t. access costs) than the current location in the recent past (i.e., in the last so-called time *epoch*). At first sight the strategy to migrate to locations that were good *in the past* might look counter-intuitive. However, the rationale is that the cost of any (even optimal offline) algorithm Alg cannot be much lower than the cost incurred by our strategy: either Alg does not migrate during a small set of epochs, but then it must incur a high access cost as well, or Alg incurs migration costs. In other words, if the distribution of the demand does not change much over time, our strategy will correctly predict the best server location and move there quickly. However, if the demand changes, *any* algorithm must have either high access or migration costs, and our algorithms are again close to optimal.

With these intuitions in mind, we can present our algorithms in more formal detail. Table I summarizes the main variables and parameters used in this and subsequent sections.

σ	sequence of requests
n	number of substrate nodes
n_1	size of largest PIP
k	number of redundant servers
size(s)	service size
β	migration cost
CoG	center of gravity
z	number of time zones
λ	request duration
p	request correlation
π	transit provider cost
Δ	diameter of PIP graph
γ	a configuration

TABLE I: Important Variables

A. Randomized Migration

Let us first consider a scenario with constant bandwidth capacities, i.e., $\omega(e) = \omega \quad \forall e \in E$ and let $\beta = \text{size}(s)/\omega$ be the corresponding migration cost. Let $\phi \in (0, 1)$ be a parameterizable threshold and c a weighting factor (another parameter of the algorithm). Both parameters ϕ and c are constants.

Algo IV.1 (Mix)**.** *The algorithm* Mix *divides time into* epochs*. In each epoch* Mix *monitors, for each node v, the cost of serving all requests from this epoch by a server kept at v. We will denote the value of such a counter at the end of step t by $\text{C}_t(v)$.* Mix *keeps the server at a single node w until step t for which $\text{C}_t(w) > c \cdot \beta$, i.e., till step t in which the counter would exceed $c \cdot \beta$ if no migration occurred. At such a step t,* Mix *migrates the server to a node u chosen uniformly at random among nodes with the property $\text{C}_t(u) \le \phi \cdot c \cdot \beta$. If there is no such node,* Mix *does not migrate the server, and the epoch ends in that round. The next epoch starts in the next round, at the beginning of which the counters are reset to zero.*

Theorem IV.2. *On expectation,* Mix *is $O(\log n)$-competitive in networks with constant bandwidth.*

Proof: Fix any epoch $\mathcal{E}$ and let β denote the migration cost. If Opt migrates the server within $\mathcal{E}$, it pays β. Otherwise it keeps it at a single node paying the value of the corresponding counter at the end of $\mathcal{E}$. By the construction of Mix, this value is at least $\phi \cdot c \cdot \beta$, and thus in either case $\text{Opt}(\mathcal{E}) = \Omega(\beta)$.

The migrations performed by Mix partition $\mathcal{E}$ into several *phases*. According to our migration strategy, the access cost of Mix in each phase is at most $c \cdot \beta$ as it moves *before* the counter exceeds this threshold. Below, we will show that the expected number of migrations within one epoch is at most H_n, where H_n is the n-th harmonic number. The maximum number of phases is then $H_n + 1$, and hence the expected value of $\text{Mix}(\mathcal{E})$ is at most $\beta \cdot H_n + c \cdot \beta \cdot (H_n + 1) = O(\beta \cdot \log n)$. This yields the competitiveness of Mix.

Let $(v_i)_{i=1}^n$ be the sequence of nodes in order their counters reach the value $c \cdot \phi \cdot \beta$ (with ties broken arbitrarily). Furthermore, fix any node v_i and let t_i be the time when its counter exceeds $c \cdot \beta$, and let K_i be the number of nodes whose counters at time t_i are at most $c \cdot \phi \cdot \beta$. Clearly $K_i \le n - i$ and those nodes are $v_{n-K_i+1}, v_{n-K_i+2}, \ldots, v_n$.

Assume that in a phase Mix keeps the server at node v_i and let T_i be the expected number of server migrations until the end of $\mathcal{E}$. If $i = n$, then the current phase is the last one in $\mathcal{E}$, and thus $T_n = 0$. Otherwise $i < n$, and at the end of this phase Mix chooses a next place for the server uniformly at random from the set $\{v_j : n - K_i + 1 \le j \le n\}$. Hence, we obtain the recursive formula $T_i = 1 + \frac{1}{K_i} \sum_{j=n-K_i+1}^{n} T_j$. By a backward induction, one sees that $T_i \le H_{n-i}$. The basis holds as $T_n = 0$ and for the inductive step, we use $T_i \le 1 + \frac{1}{K_i} \sum_{j=n-K_i+1}^{n} H_{n-j} \le 1 + \frac{1}{n-i} \sum_{j=i+1}^{n} H_{n-j} = H_{n-i}$. Thus, if Mix starts $\mathcal{E}$ with server at node v_k, the expected number of migrations in $\mathcal{E}$ is $H_{n-k} \le H_n$. ■

Note that our analysis does not rely on access costs being measured as the number of hops. Rather, the analysis (and hence also the result) is applicable to any metric which ensures that counters increase *monotonically* over time (i.e., with additional requests).

B. Deterministic Migration

For the analysis of MIX, we assumed an oblivious adversary that cannot be adaptive with respect to the random choices made by the online algorithm. We now show that even a deterministic bound can be achieved. We will again assume a weighting parameter c, a threshold $\phi \in (0, 1/2)$, $\omega(e) = \omega \ \ \forall e \in E$ and $\beta = \text{size}(s)/\omega$.

Our deterministic algorithm CEN is based on the concept of *gravity centers.*

Definition IV.3 (Gravity Center)**.** *Let $d(\cdot, \cdot)$ denote the shortest path metric on the weighted network graph G. The* gravity center *of a subset $V' \subseteq V$ of nodes is defined as the (not necessarily unique) node* $\text{CoG} = \arg\min_{v \in V'} \sum_{u \in V'} d(u, v)$*. (Ties are broken arbitrarily.)*

We can now describe CEN in detail.

Algo IV.4 (CEN)**.** CEN *divides time into* epochs *consisting of one or multiple* phases *between which* CEN *migrates. Again, we have counters* $\text{C}(v)$ *for each node v that are set to zero at the beginning of an epoch. These counters accumulate the access costs of an epoch if the server was permanently located at v. At the beginning of an epoch,* CEN *migrates the server to a gravity center of the set V. We call all nodes v, for which* $\text{C}(v) < \phi \cdot c \cdot \beta$ *at time t,* active at time t. *Assume that algorithm* CEN *is currently at some node v.* CEN *remains at this node until the node counter accumulated access costs of $c \cdot \beta$. Then, a new phase starts, and* CEN *computes the* gravity center *w, i.e., the "center" of the currently active nodes V'.* CEN *migrates to w and a new phase starts. If there is no active node left, the epoch ends.*

In order to study the competitive ratio of CEN, we exploit the property that a request always increases the counter of several nodes (namely: a constant fraction) besides the gravity center by at least a certain value (again, a constant fraction).

Lemma IV.5. *Let $\lambda \in (0, 1/2)$ be an arbitrary constant. Fix any active set V'. Let r be an arbitrary requesting node (at some step). Assume the counter at the gravity center* CoG *increased by F because of request r. Then there are at least $\lambda \cdot |V'|$ nodes from V' whose counters increased at least by $\frac{1-2\lambda}{3-2\lambda} \cdot F$.*

Proof: Assume the contrary. It means that there are at least $(1-\lambda) \cdot |V'|$ nodes from V' whose counter increase is smaller than $\alpha \cdot F$ where $\alpha = (1-2\lambda)/(3-2\lambda)$. Denote this set by V''. Nodes in V'' have a distance to the request smaller than $\alpha \cdot F$. This means that $\forall u \in V''$, $d(u, r) < \alpha \cdot F$, and therefore $\forall u, v \in V''$, $d(u, v) < 2\alpha \cdot F$, i.e., the diameter of the set V'' is relatively small. On the other hand, the distance between the request r and the gravity center is $d(\text{CoG}, r) = F$.

Now let ξ be any node of V''. We show that ξ would be a better candidate for the gravity center than CoG. Given that $d(\text{CoG}, r) = F$, that $d(u, r) < \alpha \cdot F$ for all $u \in V''$, (and that $d(\xi, r) < \alpha \cdot F$), using triangle inequality, we obtain the following inequalities.

1) For any $u \in V''$, it holds that $d(\text{CoG}, u) \geq d(\text{CoG}, r) - d(u, r) > (1-\alpha) \cdot F$.
2) For any $u \in V' \setminus V''$, it holds that $d(\xi, u) < d(\xi, r) + d(r, \text{CoG}) + d(\text{CoG}, u) < (1+\alpha) \cdot F + d(\text{CoG}, u)$.

Thus,

$$\sum_{u \in V'} d(\text{CoG}, u) = \sum_{u \in V''} d(\text{CoG}, u) + \sum_{u \in V' \setminus V''} d(\text{CoG}, u)$$
$$> (1-\alpha) \cdot |V''| \cdot F + \sum_{u \in V' \setminus V''} d(\text{CoG}, u)$$

and

$$\sum_{u \in V'} d(\xi, u) = \sum_{u \in V''} d(\xi, u) + \sum_{u \in V' \setminus V''} d(\xi, u)$$
$$< 2\alpha \cdot |V''| \cdot F + |V' \setminus V''| \cdot (1+\alpha) \cdot F$$
$$+ \sum_{u \in V' \setminus V''} d(\text{CoG}, u)$$

Finally, note that by definition of V'', $|V' \setminus V''| \leq \lambda \cdot |V'| \leq \frac{\lambda}{1-\lambda} \cdot |V''|$. Thus, $\sum_{u \in V'} d(\text{CoG}, u) - \sum_{u \in V'} d(\xi, u) > (1 - 3\alpha - \frac{\lambda}{1-\lambda}(1+\alpha)) \cdot |V''| \cdot F \geq 0$, which means that ξ is a better gravity center than CoG: a contradiction. ■

From Lemma IV.5 it follows that when the counter at the gravity center exceeds a given threshold, the counter of many nodes besides the center must be high as well.

Lemma IV.6. *Fix any active set V', any threshold θ, and a constant $\lambda \in (0, 1/2)$. When the counter at the gravity center* CoG *exceeds θ, then there exists $V'' \subseteq V'$, such that $|V''| \geq \lambda/2 \cdot |V'|$ and the counter at any node $v \in V''$ is at least $\frac{1-2\lambda}{3-2\lambda} \cdot \frac{\lambda}{2} \cdot \theta$.*

Proof: Assume the contrary, i.e., there exists $V'' \subseteq V'$, such that $|V''| > (1-\lambda/2) \cdot |V'|$, and for all $v \in V''$, the counter at v is smaller than $\phi \cdot \theta$, where $\phi = \frac{1-2\lambda}{3-2\lambda} \cdot \frac{\lambda}{2}$. In consequence, $\sum_{v \in V''} \text{C}(v) < |V''| \cdot \phi \cdot \theta \leq |V'| \cdot \phi \cdot \theta$. On the other hand, by Lemma IV.5, each time the counter $\text{C}(\text{CoG})$ increases by F, at least $\lambda \cdot |V'|$ counters from set V' (and hence at least $\lambda/2 \cdot |V'|$ counters from set V'', since $|V' \setminus V''| \leq \lambda/2 \cdot |V'|$) increase by $\frac{1-2\lambda}{3-2\lambda} \cdot F$. Thus, the sum of counters from V'' increases at least by $\phi \cdot |V'| \cdot F$. Therefore, when $\text{C}(\text{CoG}) \geq \theta$, then $\sum_{v \in V''} \text{C}(v) \geq |V'| \cdot \phi \cdot \theta$, which is a contradiction. ■

For the competitive ratio, we therefore have the following result.

Theorem IV.7. CEN *is $O(\log n)$-competitive in constant bandwidth scenarios.*

Proof: First, we consider the cost of the optimal offline algorithm. If OPT migrates in an epoch, it has costs β. Otherwise, due to the definition of CEN, as there are no active nodes left at the end of an epoch, the access costs of any node is at least $\phi \cdot c \cdot \beta = \Omega(\beta)$. Regarding CEN, its access cost in each phase is at most $c \cdot \beta$, and it remains to study the maximum number of phases per epoch. For any CEN's parameter $\phi \in (0, 1/2)$, there exists a constant λ, such that $\phi = \frac{1-2\lambda}{3-2\lambda} \cdot \frac{\lambda}{2}$. By the algorithm definition, within a single phase CEN remains at a gravity center of the set of nodes that are active at the beginning of the phase. As the counter at the gravity center increases by $c \cdot \beta$, by Lemma IV.6, in each phase the counters at a fraction of at least $\lambda/2$ of the active nodes increase by $\frac{1-2\lambda}{3-2\lambda} \cdot c \cdot \beta = \phi \cdot c \cdot \beta$, which means that they cease to be active. Thus, the number of active nodes is reduced at least by a factor $1 - \lambda/2$. Therefore, there are at most $\log n / \log(1-\lambda/2) = O(\log n)$ many phases per epoch, and the claim follows. ■

C. Lower Bound

It can be shown that the competitive ratios achieved by MIX and CEN are close to optimal.

Theorem IV.8. *There are graphs on which the competitive ratio of any randomized or deterministic algorithm is at least* $\Omega(\log n / \log \log n)$.

Proof: To prove the theorem, we consider an instance of the *online function tracking problem* [30] with linear penalties [7]. In this variant of the problem, an algorithm observes a sequence of numbers and wants to represent their values by communicating a special "representative" value y. An algorithm (observer) is charged both for communication (i.e., updating the counter) and for inaccuracies of the representation. More concretely, in each step t, the following happens: (i) a number x_t is revealed to the observer, (ii) the observer may pay fixed amount β and change the value y to an arbitrarily chosen number, and (iii) the observer pays the difference between the value stored in y and x_t. Theorem 6 of [7] states that no online algorithm for this problem (not even a randomized one) can achieve a competitive ratio smaller than $\Omega(\log \beta / \log \log \beta)$. The lower bound construction uses only integer values $x_t \in \{0, \ldots, \beta - 1\}$.

To show the lower bound for the server migration problem, we can simply consider a fixed linear graph (i.e., a *chain*) of $n = \beta$ nodes, numbered from 0 to $n-1$. On such a graph, there is a one-to-one correspondence between the server migration problem and function tracking with linear penalties: observing a value x_t corresponds to an access at x_t, changing value of y to h corresponds to moving a server to node h, and paying for the inaccuracy between h and x_t corresponds to paying for request x_t with a server at h. Thus, the lower bound $\Omega(\log \beta / \log \log \beta) = \Omega(\log n / \log \log n)$ of [7] also applies to the server migration problem.

■

As discussed in the related work section, our problem can be seen as a special instance of the *n-point uniform space metrical task system* (where all distances are equal), and it is tempting to try to port the $\Omega(\log n)$ lower bound also to our setting (see, e.g., [19]). However, note that due to the topological constraints of the access costs, a better ratio is potentially possible for our service migration problem. The question whether online algorithms with a competitive ratio $o(\log n)$ exist hence remains a challenging open problem.

D. Remark

The adversarial model for MIX and CEN is different, and one has to be careful when comparing the competitive ratios. The bound for MIX only holds against oblivious adversaries, and we expect the center of gravity approach to perform better in worst-case scenarios with adaptive adversaries. Our simulations show that on average, however, the two strategies often yield a similar performance.

Also note that both MIX and CEN are quite general with respect to the measure of access costs, i.e., the derived bounds hold for arbitrary latency functions on the links in case of MIX. In case of CEN, the access costs must fulfill the triangle inequality.

V. OPTIMAL OFFLINE ALGORITHM OPT

In the competitive analysis of our online algorithms we argued about an optimal offline algorithm to which we compare our costs. However, this algorithm was hypothetical as there was no need to find or describe the offline algorithm explicitly: all we needed was a good lower bound on its cost.

Still, while the decisions when and where to migrate servers typically need to be done *online*, i.e., without the knowledge of future requests, there can be situations where it is interesting to study which migration pattern would have been optimal *at hindsight*. For example, if it is known that the requests follow a regular pattern (e.g., a periodic pattern per day or week), it can make sense to compute an optimal migration strategy offline and apply it in the future. Another reason for designing optimal offline algorithms explicitly is that an optimal solution is required to compute the competitive ratio in our simulations.

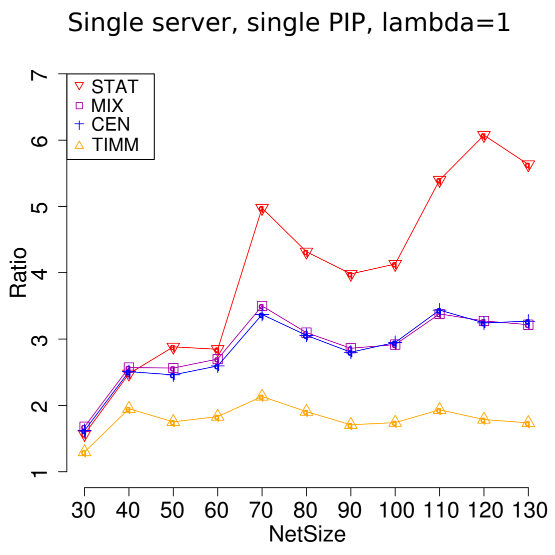
In the following, we sketch an optimal offline algorithm for the service migration problem. It is based on a standard dynamic programming approach. (See [14] for an introduction.) As we will see in Section VII, using similar techniques, optimal offline algorithms can be computed in polynomial time in much more general settings as well. We will hence postpone a detailed discussion to the corresponding sections.

Our algorithm exploits the fact that migration exhibits an optimal substructure property: Given that at time t, the server is located at a node u, the most cost-efficient migration path that leads to this configuration consists solely of optimal sub-paths. That is, if a cost minimizing path to node u at time t leads over a node v at time $t' < t$, then there cannot be a cheaper migration sub-path that leads to v at time t' than the corresponding sub-path.

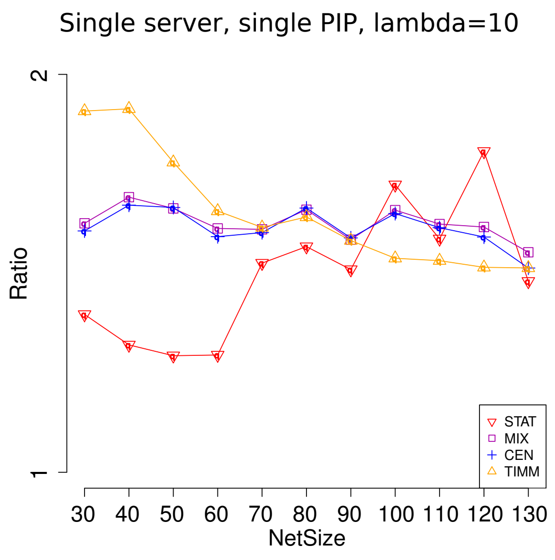
OPT fills out a matrix $opt[\text{time}][\text{node}]$ where $opt[t][v]$ contains the cost of the minimal migration path that leads to a configuration where the server satisfies the requests of time t from node v. Assume that initially (i.e., $t = 0$), the service is located at node v_0. Thus, initially, $opt[0][v_0] = 0$ and $opt[0][u] = \infty$ for all nodes $u \neq v_0$.

For $t > 0$, we find the optimal values $opt[t][u]$ by considering the optimal migration paths to any node v at time $t-1$, and adding the migration cost from v to u. That is,

$$opt[t][u] = \min_{v \in V}\{opt[t-1][v] + Cost_{\text{mig}}(v, u) + \textstyle\sum_{w \in \sigma_t} Cost_{\text{acc}}(w, u)\}$$



(a) $\lambda = 1$



(b) $\lambda = 10$

Fig. 1: Competitive ratio as a function of network size, averaged over all $p \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

where we assume that $Cost_{\text{acc}}$ includes the first (wireless) hop of the request from the terminal to the substrate network, and where $Cost_{\text{mig}}(v, v) = 0$ for all v.

We have the following runtime result.

Theorem V.1. *The optimal offline migration policy* OPT *can be computed in $O(n^3 + n^2 \sum_{t \in \Gamma} |\sigma_t|)$ time, where Γ is the set of rounds in which events occur.*

Proof: Note that we can constrain ourselves to optimal offline algorithms where migration will only take place in "active" rounds Γ with at least one request. This is useful in case of sparse sequences with few requests. The $opt[\cdot][\cdot]$-matrix contains $|\Gamma| \cdot n$ entries. In order to compute a matrix entry, we need to consider each node $v \in A$ from which a migration can originate; for each such node, the access cost from all the requests in σ_t need to be computed. Both the shortest access paths and the migration costs can be looked up in a pre-computed table (pre-computation in time at most $O(n^3)$, e.g., by *Floyd-Warshall's algorithm*) and require a constant number of operations only. ■

VI. Simulations

We complement our worst-case analysis with simulations under a specific use-case. For simplicity, we also use the term "competitive ratio" in this section. Note however that due to the specific scenario, the ratio now has a different semantics and simply refers to the *cost ratio* between the online and offline algorithm.

A. Scenario

We study a simplified *moving sun* (or *time zone*) scenario. This scenario is relatively general in the sense that it allows to change the correlation of the requests as well as the degree of dynamics, and hence lets us focus on the main factors influencing the performance of our service migration algorithms. Since geographic and topological distances correlate only to a certain extent in the real world, we will concentrate on an on-off model rather than a model where terminals "move" along the substrate topology.

Scenario VI.1 (Time Zones Scenario)**.** *The time zone scenario models an access pattern that can result, e.g., from global daytime effects. A* current *zone is chosen round-robin from the set of z available zones ($z = 16$ in our experiments). At each time t, new requests originate from nodes chosen uniformly at random from the respective current zone, until they represent a fraction p of all requests. The remaining request locations are chosen uniformly at random from all nodes. Request durations follow an exponentially distributed sojourn time with parameter λ.*

B. Set-Up

As a substrate topology for our experiments we use connected *Erdös-Rényi* random graphs with between 30 to 130 nodes, and an average 1.4 links per node. The graphs are partitioned in eight zones of equal size.

Link bandwidths are chosen at random (either T1 (1.544 Mbit/s) or T2 (6.312 Mbit/s)), and the server size is 2048MB. If not stated otherwise, we assume that $c = 1$ and that β equals the server size divided by the average bandwidth. To provide an incentive for migration, inter-zone links are weighted with a latency of 100 units whereas other links bear a unit weight. Experiments ran for 400 requests, and our results are averaged over five repetitions. All compared algorithms start at the center of the (weighted) graph.

To take into account that larger networks typically also have a larger demand, we assume that the number of requests per round is one fifth of the network size. Note that the runtime of the optimal offline algorithm and hence the computation of the competitive ratio is expensive in large networks; therefore, the scale of our experiments is typically limited.

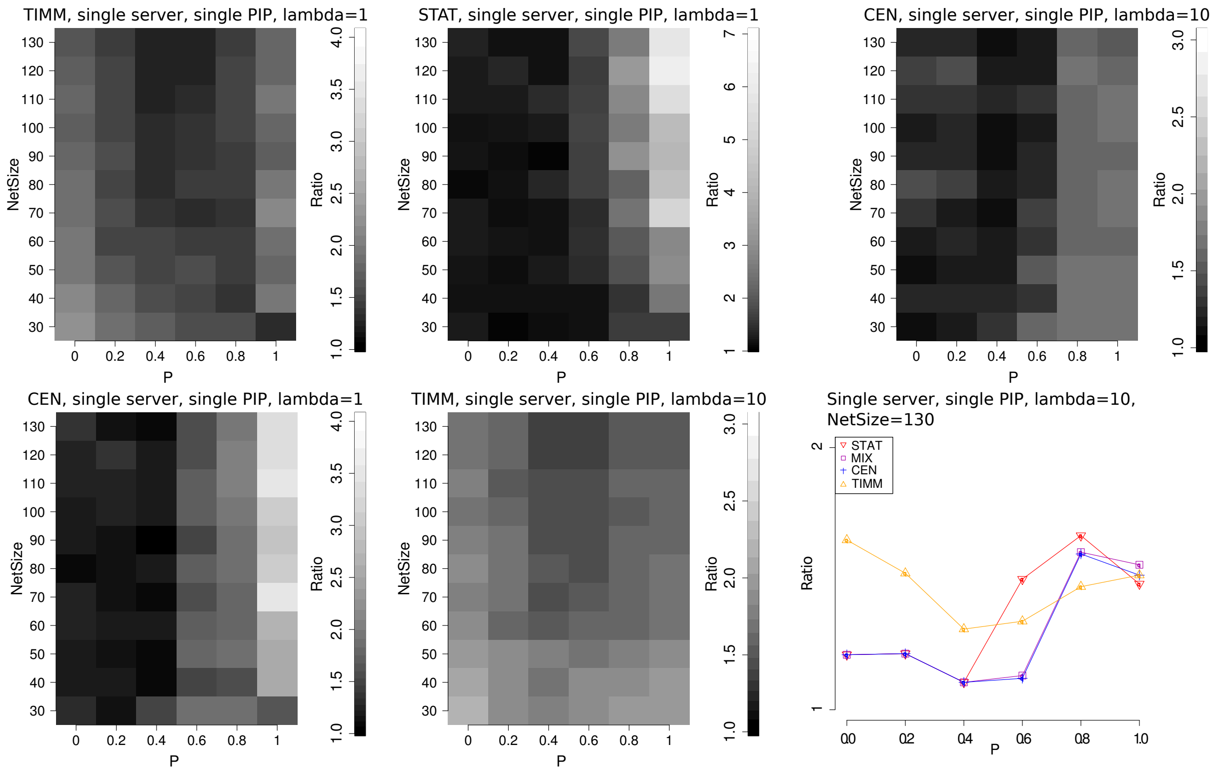


Fig. 2: Results for different parameter sets.

C. *Reference Algorithms*

Our main yardstick to evaluate the performance of MIX and CEN is the optimal offline algorithm (i.e., the competitive ratio). However, it is interesting to compare our algorithms to two additional canonic migration strategies: STAT and TIMM.

The STAT algorithm is very simple: it does not exploit the possibility to migrate and just places the service at the static center of the (weighted) topology. STAT stands for *static*. The second algorithm we consider is called TIMM, which stands for *TIMed Migration*. TIMM has a regular migration schedule: it follows the official cycle of changing zones by choosing a destination uniformly at random amongst all nodes of the new zone.

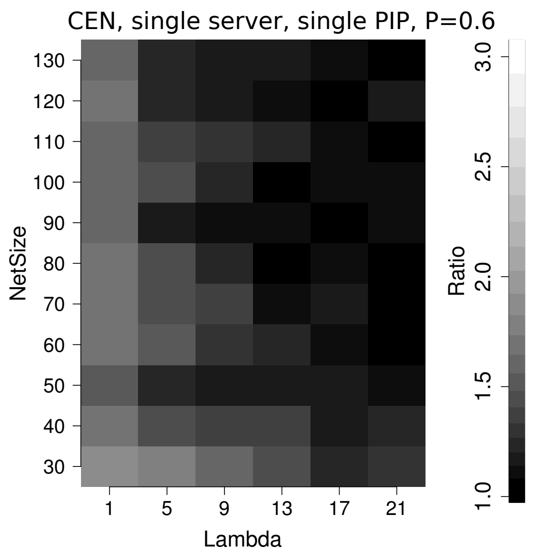
D. *Results*

We first study the competitive ratio as a function of the number of nodes under low (i.e., $\lambda = 1$ in Figure 1a) and high (i.e., $\lambda = 10$ in Figure 1b) dynamics. We can see that TIMM benefits when λ is small as the zone can be estimated relatively accurately. The competitive ratio of CEN and MIX is similar but higher. The ratio grows moderately for larger networks, which qualitatively corresponds to our formal worst-case bounds. STAT exhibits the highest ratio. As expected, it increases with the network size (i.e., with increasing diameter). For larger λ, CEN and MIX are relatively better: the migration cost can be amortized. TIMM on the other hand suffers from the reduced predictability, migrating ahead of the shift in demand. Moreover, high values of λ and large networks allow the offline algorithm OPT to optimize migrations better. Due to the more frequent migrations, the competitive ratios are generally lower for high λ values.

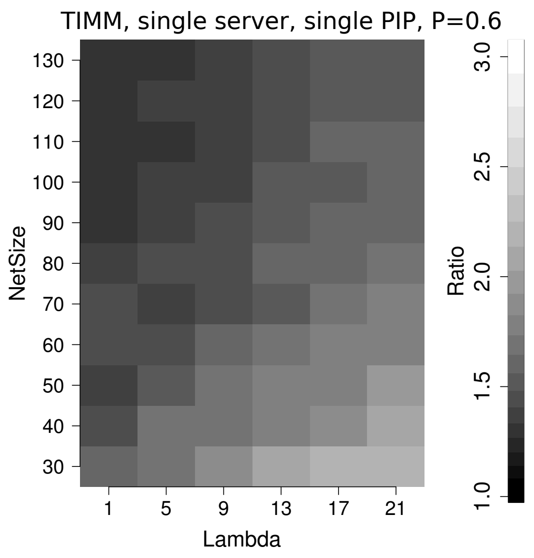
Generally, we find that the performance of CEN and MIX is often qualitatively and quantitatively similar. We therefore omit MIX plots for most of the following experiments. Moreover, in the following, we will focus on the more challenging case $\lambda = 1$. Plots for alternative parameter settings are only presented to highlight specific properties.

Figure 2 shows an overview of the competitive ratios as a function of the request correlation p. The figure indicates that all algorithms suffer from a full correlation of $p = 1$. However, the reasons are different. At low p values, economic migration destinations are scarce and not obvious. CEN and MIX therefore imitate STAT and hardly migrate for low p. TIMM migrates in any case, and incurs high costs. For $p \in [0.4, 0.6]$, useful migration destinations become available, and choosing the 'wrong' zone is not yet dire. This yields good ratios for TIMM and CEN/MIX. With $p = 1$ however, TIMM, CEN and MIX suffer from a bad choice (the prior running ahead, the latter delayed by slowly filling counters). As in the previous plots, $\lambda = 10$ allows CEN/MIX to amortize the late migration, while the minimum of TIMM's ratio curve shifts to the left. As expected, STAT yields the relatively best results with low correlation and small network sizes.

Figures 3a and 3b study the influence of λ. We see an inverse effect on CEN and TIMM. Due to the limited amount of requests and our method of choosing request origins, TIMM's ratio increase however remains limited. Very high λ values have the effect of omitted zones. If requests of a zone remain active over the whole period of its successor, requests will



(a) Cen



(b) Timm

Fig. 3: Competitive ratio as a function of λ and network size for $p = 0.6$.

jump this zone and catch up with Timm.

Overall, Timm performs well for moderate p and even on average over all p for low λ. However, Cen/Mix improve for higher λ and adapt to different correlation values p well. More specifically, they do not migrate in situations where Stat shows good performance. They may therefore be preferable in scenarios with moderate to high λ and low to moderate request correlation, adapting to occasional spikes in p. Effectively, being adaptive they may be considered a suitable initial choice for new services with yet insufficiently known usage patterns.

VII. Extensions

The algorithmic principles of Mix and Cen can serve as a basis for more involved models. This section initiates the discussion of such extensions. Concretely, we will study scenarios where a service can be migrated across provider boundaries (Section VII-A), where a service is realized with multiple servers (Section VII-B), and where substrate resources are more heterogenous (Section VII-C). We understand our algorithms and bounds in this section as a first look at the more complex models. More research is needed to gain insights into the optimal tradeoffs arising in these scenarios.

A. Multiple Providers

The flexibility offered by network virtualization is not limited to a single PIP. Rather, a Virtual Network Provider may have contracts with multiple infrastructure providers, and provision a service across PIP boundaries. In the following, we sketch how to extend our algorithms to multiple provider scenarios. In particular, we assume that migrating a server across a PIP entails a fixed "roaming" cost π for each transit. Since we assume that a PIP typically does not reveal its internal resource structure, we seek to come up with migration algorithms that pose minimal requirements on the knowledge of a PIP topology.

We can extend the deterministic algorithm Cen to multi-PIP scenarios as follows. (The extension for the randomized algorithm Mix is analogous and omitted here.) We consider x PIPs, and assume that migration inside a PIP costs β, that access costs are the number of hops, and that migrating across providers costs π per crossed PIP. We will concentrate on the more realistic case where $\pi \geq \beta$.[1]

It is sometimes useful to think of the *PIP graph*, the graph where all the nodes of one PIP form one vertex and two PIPs are connected if there is a connection between nodes of the respective PIPs in the substrate graph. In particular, we will refer to the *diameter of the PIP graph*, the largest number PIP boundaries to be crossed on a shortest migration path, by Δ.

Algo VII.1 (MultiPIP). *The algorithm* MultiPIP *divides time into three types of* epochs*: a* huge epoch *consists of multiple* large epochs*, and a large epoch consists of* $\psi\lceil\pi/\beta\rceil$ small epochs*, for some constant* ψ*. Again, we use counters* $\mathrm{C}(u)$ *to accumulate the access costs of a node* u *during a small epoch; in addition, a counter* $\mathrm{C}_L(u)$ *is used to accumulate access costs during a large epoch. In the beginning, all PIPs are set to* active*. At the beginning of a small epoch, the* $\mathrm{C}(u)$ *values are set to zero for all nodes within the current PIP.* MultiPIP *then monitors, for each node* u*, the cost of serving all requests from this small epoch by a server kept at* u*.* MultiPIP *leaves the server at a single node* u *until* $\mathrm{C}(u)$ *reaches* β*. In this case,* MultiPIP *migrates the server to a node* v *which constitutes the center of gravity among the* active *nodes of the current PIP, i.e., the nodes* w *of the current PIP for which it still holds that* $\mathrm{C}(w) < \beta/\psi$*. If there is no active node left within the current PIP, a small epoch ends in that round; the next small epoch starts in the next round. After* $\psi\lceil\pi/\beta\rceil$ *small epochs, a new large epoch starts, and all*

[1]If the total migration cost (over multiple providers) is in the order of β, our single PIP algorithms could be applied without taking into account transit costs.

nodes u in the network with $\mathrm{C}_L(u) \geq \pi/\psi$ become inactive with respect to the large epoch. Among all remaining active nodes of the large epoch, MULTIPIP *determines their center of gravity and moves the server to the corresponding PIP, and a new large epoch begins. Otherwise, if there is no PIP left that contains active nodes, the server stays where it is, and a new huge epoch starts.*

The following theorem can be shown along the lines of the corresponding single provider proof (see [1] for details).

Theorem VII.2. *For a sufficiently large value of constant ψ,* MULTIPIP *is $O(\log n \cdot (\log n_1 + \Delta))$-competitive in networks with constant bandwidth and x PIPs, where n_1 is the size of the largest PIP, and Δ is the diameter of the PIP graph.*

Note that MULTIPIP requires knowledge of the topologies of the different PIPs to compute the gravity centers. However, this information may not be available if PIPs conceal their network (typically a business secret). Thus, what can be optimized will depend on the specific context and information shared by the PIPs. Our algorithms must be adapted to make use of the potentially aggregated or estimated costs. Nevertheless, we expect the approach to be relatively robust to approximate information. For example, we expect that pragmatic implementations that move the service, e.g., to the PIP which lies "at the center" of the active providers, yield good results and justify the validity of concept and analysis. Moreover, it turns out that MIX can be generalized with less assumptions on the infrastructure topology, allowing for a higher autonomy on the PIP level. [1]

B. Redundant Servers

A natural way to generalize our algorithms to a scenario where a service is realized with up to $k > 1$ redundant servers, is to use *configurations*.

Definition VII.3 (Configuration)**.** *A* configuration γ *describes, for each server, whether it is* not in use, inactive*, or* active*. In case of inactive and active servers, γ specifies where—i.e., on which node—the server is located.*

The following algorithm CONF is an extension of MIX to multi-server scenarios. (The extension for CEN is analogous.) For simplicity, we will assume that migration is cheap, i.e., that $\beta < c$; the case where $\beta > c$ is similar, see [2].

Algo VII.4 (CONF)**.** CONF *uses a counter* $\mathrm{C}(\gamma)$ *for each configuration γ. Time is divided into* epochs*. In each epoch* CONF *monitors, for each configuration γ, the cost of serving all requests from this epoch by servers kept in configuration γ, including the access costs (latency plus induced load) of the requests, the server running costs, and possible creation costs.* CONF *stores this cost in* $\mathrm{C}(\gamma)$*. The servers are kept in a given configuration $\widehat{\gamma}$ until* $\mathrm{C}(\widehat{\gamma})$ *reaches $k \cdot c$, where k is the maximal number of servers. In this case,* CONF *changes to a configuration $\widehat{\gamma}'$ chosen uniformly at random among configurations with the property* $\mathrm{C}(\gamma) < k \cdot c$*. If there is no such configuration left, the epoch ends in that round; the next epoch starts in the next round and the counters* $\mathrm{C}(\gamma)$ *are reset to zero.*

Although a certain bound on the competitive ratio of CONF can be obtained (see [2]), the algorithm needs to be optimized in many respects. For instance, it can make sense to switch between "close" (with respect to costs) configurations only, or to deterministically switch to the configuration with the lowest counter. More importantly, the complexity of this algorithm is high for a large number of servers k. In order to speed up the computations, *local search* heuristics could be employed where only one server of the configuration changes state per epoch. A detailed discussion is left for future research.

C. Heterogenous Resources

Finally, let us have a look at a scenario with heterogenous resources, i.e., where the different links have different bandwidths. In this scenario, a service will be migrated along the path with the highest bandwidth, as the service interruption cost is given by the server size and the minimal available migration bandwidth.

There is a simple generalization of our results in Section IV to arbitrary bandwidths: without major modifications of CEN and MIX we have the following results.

Theorem VII.5. MIX *and* CEN *are $O(\mu \cdot \log n)$-competitive in general networks, where $\mu = \max_{e,e' \in E} \omega(e)/\omega(e')$.*

Proof: In networks with arbitrary and heterogenous bandwidths, the algorithms can simply be adopted in such a way that we define the migration cost β to be $\mathrm{size}(s)/\max_e \omega(e)$. As β is a lower bound on the actual migration cost, the lower bounds on OPT (given as a function of β) remain intact. On the other hand, the actual cost of our algorithm is underestimated at most by a factor of μ. ■

To deal with heterogenous bandwidths more efficiently, we may extend MIX (and similarly: CEN) by a *hierarchical clustering* approach that classifies edges in different classes depending on their bandwidth. We leave the study of this approach for future research, but it remains to remark that (after a clustering) the heterogenous bandwidth problem can also be modeled as an instance of a *Metrical Task System* on a *hierarchically separable tree metric*. Although good competitive ratios are achieved (in the order of $O(\log n \cdot \log \log n)$ [17]), the corresponding randomized algorithms are quite complex and thus to some extent impractical.

D. Algorithmic Offline Framework

Regarding our offline algorithm, there exists a simple generalization for all these models as well. The idea of a generic optimal offline algorithm GOPT is to use the concept of configurations (cf Definition VII.3). Recall that given a configuration γ, access costs $\mathrm{Cost}_{\mathrm{acc}}$, migration costs $\mathrm{Cost}_{\mathrm{mig}}$, and the running costs $\mathrm{Cost}_{\mathrm{run}}$ over time can be computed.

GOPT exploits optimal substructure properties on the configuration level: Given that at time t, the k servers are in a configuration γ, then the most cost-efficient *path* (migrations, activation and deactivation of servers, creation, etc.) that leads to this configuration consists solely of optimal sub-paths. That is, if a cost minimizing path to configuration γ at time t leads over a configuration γ' at time $t' < t$, then there cannot be a cheaper migration sub-path that leads to γ' at time t' than the corresponding sub-path.

GOPT essentially fills out a matrix $opt[\text{time}][\text{configuration}]$ where $opt[t][\gamma]$ contains the cost of the minimal path that leads to a configuration where the servers satisfy the requests of time t in a configuration γ. For $t > 0$, we find the optimal values $opt[t][\gamma]$ by considering the optimal paths to any configuration γ' at time $t-1$, and adding the migration cost from γ' to γ. That is, the optimal cost to arrive at a configuration with servers at γ at time t is equal to

$$\min_{\gamma'}\{opt[t-1][\gamma'] + \text{Cost}(\gamma' \to \gamma) + \text{Cost}_{\text{run}}(\gamma) + \textstyle\sum_{v \in \sigma_t} \text{Cost}_{\text{acc}}(v, \gamma)\}$$

where we assume that Cost_{acc} includes the first (wireless) hop of the request from the terminal to the substrate network, and where $\text{Cost}(\gamma \to \gamma) = 0$.

Observe that the runtime of GOPT is relatively low for a small number of servers, i.e., any scenario with a constant upper bound on the number of usable servers can be computed in polynomial time. However, for a non-constant number of servers, clustering or sampling heuristics may be needed to speed up the computations; only approximative solutions can be found.

VIII. Service Migration Demonstrator

To round off and complement our theoretical study, we report on our network virtualization prototype architecture [29] which allows to migrate services. In particular, the prototype includes a migration demonstrator.

The demonstrator performs two types of migrations: (1) The migration of a virtual video streaming server (technically: a virtual machine) closer to the users; the migration decisions are based on the center-of-gravity algorithm presented in this article (namely CEN). For example, these migrations may visualize a "'move-with-the-sun" scenario where the server moves along with the users. (2) The migration of entire virtual networks (VNet) which are already embedded in the substrate network and may be moved away ("move-with-the-moon") to free up resources for the streaming server. These VNets do not have any latency requirements towards users, and embeddings and migrations are computed using mathematical programming (namely, Mixed Integer Programs [28]).

Accordingly, the demonstrator consists of the following two parts:

a) The video demonstrator: This part demonstrates how a migrating streaming server can improve QoS for (mobile) terminals (one possible use case for our network virtualization architecture). It consists of a web interface (see Figure 4) for control and three virtual network (VNet) topologies. Each of these topologies contains a video streaming server VNode (`server`) and three access point VNodes (`accesspoint{1,2,3}`) in fixed locations. These access points connect video streaming clients to the server (zero to three for each access point). Each topology mandates co-location of the streaming server with a different access point.

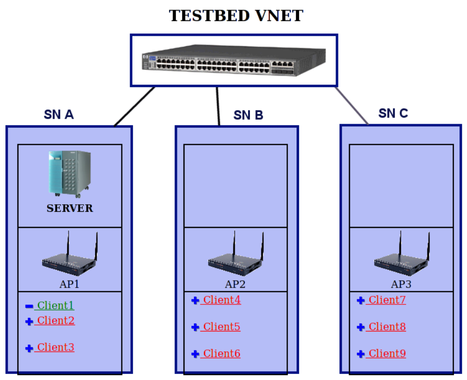


Fig. 4: Video Demonstrator web interface.

b) The topology demonstrator: This part consists of several pre-defined VNet topologies that can be embedded through a web interface. These topologies are independent of the video demonstrator virtual network, but they share the same substrate (thus competing for the same resources). The topology demonstrator's VNets are priorized *lower* than the video demonstrator VNet.

Depending on client load the video demonstrator's streaming server will migrate to be co-located with the center-of-gravity access points. At the same time topology demonstrator VNets may be embedded on the same substrate. In case of local resource shortages[2] the topology demonstrator's virtual machines will be migrated away to accommodate the video demonstrator's VNodes.

A. Background on Prototype

Our network virtualization prototype is run on two separate testbed environments, one at TU Berlin (the *Routerlab*) and one at NTT DoCoMo Eurolabs, Munich. Both testbeds have a Cisco 4500 series switch carrying both virtual network *data plane* VLANs and *testbed management* VLANs.

Substrate nodes are Sun X4150 machines hosting both virtual network nodes and the virtual machines running physical infrastructure provider management software. Substrate nodes, management nodes and virtual nodes run Linux (`Ubuntu 8.04`). Virtual nodes are *Xen* or *KVM* virtual machines running on the substrate nodes.

The substrate resource allocation for the VNet embeddings (CPU and memory resources on virtual machine hosts, and network links for connections) is computed using a *mixed integer program* [28]; it also supports migrations. Once the resource allocation is determined, the physical infrastructure provider creates virtual machines, establishes the links between them and hands control over to the customer (e.g. by providing console access to the virtual nodes).

Each physical infrastructure provider keeps a MySQL database with information about its substrate topology and embedded virtual networks. The database holds a representation of the substrate topology (the *Underlay (UL) Graph*), and

[2]The substrate description contains exactly enough space for all topology demonstrator VNodes, `accesspoint1,2,3` and *one* instance of the `server` VNode, but there are enough extra resources to allow for a brief oversubscription during migration.

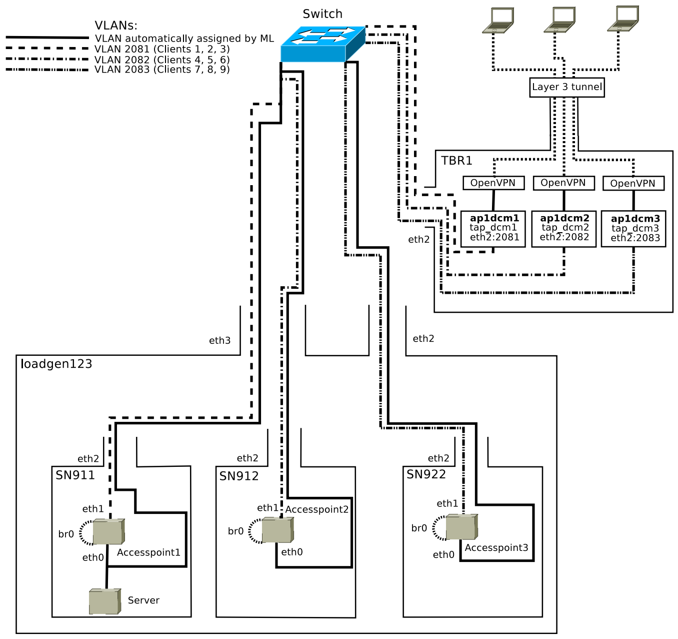


Fig. 5: Video Demonstrator overview schematic. One VLAN connects all three access points and the streaming server. Each triple of clients shares a dedicated VLAN (created manually). The access points bridge the client VLANs into the virtual network.

for each embedded virtual network there is an *Overlay (OL) Graph* describing the virtual network topology and a *Mapping Layer (ML) Graph* describing the locations of the virtual network's components on the substrate hosting them.

B. Implementation

The Video Demonstrator includes three components: a range of backend scripts behind the video demonstrator's web interface, a video streaming server on the *server* virtual node, and a multi-layered VPN tunnel architecture to connect three physical machines, each hosting three streaming clients to the virtual network (see Figure 5 for a detailed schematic.).

The streaming server uses VLC's[3] stateless UDP streaming protocol to send a video stream to a total of nine clients evenly distributed across the aforementioned physical machines. Said machines in turn display the video stream. Individual clients are enabled/disabled through the video demonstrator's web interface. Video is continuously streamed to all clients, with streams to disabled clients being blocked by `iptables`[4] rules on the *server* VNode.

Upon each change in client status the web interface's backend will recompute the center of gravity (cf Definition IV.3). If it changes it will issue a topology modification request (to the infrastructure provider hosting the video demonstrator VNet) that co-locates the server with the access point at the new center of gravity. The infrastructure will then recompute the embedding and migrate the *server* VNode accordingly.

[3]`http://www.videolan.org`
[4]`http://www.netfilter.org`

IX. Conclusion

At the heart of network virtualization lies the ability to react to changing environments in a flexible fashion. In order to optimally exploit the benefits from virtualization, cost-aware migration algorithms need to be designed which this is typically difficult as future demand is hard to predict. We believe that competitive analysis is an important tool to devise and understand such online algorithms.

This article studied the cost-benefit tradeoff of online migration in a system supported by network virtualization, and compared our system to a setting without migration. We derived the first migration algorithms, which are competitive even in the worst-case.

We understand our work as a first step towards a better understanding of competitive virtual service migration, and there are several interesting directions for future research. For instance, we have so far only sketched extensions to multi-provider scenarios or scenarios with redundant servers realizing a service. The optimal tradeoffs remain unclear.

Acknowledgments

We would like to thank our colleagues in the virtu projects for many fruitful discussions; in particular: Dan Jurca, Wolfgang Kellerer, Ashiq Khan, Kazuyuki Kozu, and Joerg Widmer. Special thanks go to Ernesto Abarca who was a great help during the prototype implementation.

Marcin Bienkowski was supported by MNiSW grant number N N206 368839, 2010-2013. Part of this work was performed within the Virtu project, funded by NTT DoCoMo Euro-Labs, and the Collaborative Networking project funded by Deutsche Telekom AG.

References

[1] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid. Online strategies for intra and inter provider service migration in virtual networks. In *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, IPTcomm '11, pages 10:1–10:11, New York, NY, USA, 2011. ACM.
[2] D. Arora, A. Feldmann, G. Schaffrath, and S. Schmid. On the benefit of virtualization: Strategies for flexible server allocation. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, Boston, USA, March 2011.
[3] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proc. 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–276, 2011.
[4] A. Belbekkouche, M. M. Hasan, and A. Karmouch. Resource discovery and allocation in network virtualization. *Communications Surveys Tutorials, IEEE*, 14(4):1114 –1128, quarter 2012.
[5] M. Bienkowski. Migrating and replicating data in networks. *Computer Science - Research and Development*, pages 169–179, 2011.
[6] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer. Competitive analysis for service migration in vnets. In *Proc. ACM SIGCOMM VISA Workshop*, pages 17–24, New Delhi, India, August 2010. ACM.
[7] M. Bienkowski and S. Schmid. Online function tracking with generalized penalties. In *Proc. 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 359–370, 2010.
[8] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
[9] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.
[10] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schioeberg. Live wide-area migration of virtual machines including local persistent state. In *Proc. VEE*, pages 169–179, 2007.

[11] D. Breitgand, G. Kutiel, and D. Raz. Cost-aware live migration of services in the cloud. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, pages 3–3, 2011.

[12] K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM*, pages 783–791, 2009.

[13] M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Elsevier Computer Networks*, 54(5):862–876, 2010.

[14] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 2001.

[15] G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and deterministic embeddings of virtual networks. In *Proc. 13th International Conference on Distributed Computing and Networking (ICDCN)*, pages 106–121, 2012.

[16] J. Fan and M. H. Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *Proc. IEEE INFOCOM*, pages 641–652, 2006.

[17] A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

[18] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song. Enhancing dynamic cloud-based services using network virtualization. *SIGCOMM Comput. Commun. Rev.*, 40(1):67–74, 2010.

[19] S. Irani and S. Seiden. Randomized algorithms for metrical task systems. *Theor. Comput. Sci.*, 194:163–182, 1998.

[20] M. Lin, Z. Liu, A. Wierman, and L. Andrew. Online algorithms for geographical load balancing. In *Proc. IEEE International Green Computing Conference (IGCC)*, pages 1–10, 2012.

[21] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. ACM SIGCOMM VISA*, pages 81–88, 2009.

[22] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. In *WUCSE-2006-35, Washington University*, 2006.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[24] B. Monien and H. Sudborough. Embedding one interconnection network in another. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. Syslo, editors, *Computational Graph Theory*, volume 7 of *Computing Supplementum*, pages 257–282. Springer Vienna, 1990.

[25] K.-M. Park and C.-K. Kim. A framework for virtual network embedding in wireless networks. In *Proc. 4th International Conference on Future Internet Technologies (CFI)*, pages 5–7, 2009.

[26] R. Potter and A. Nakao. Mobitopolo: A portable infrastructure to facilitate flexible deployment and migration of distributed applications with virtual topologies. In *Proc. ACM SIGCOMM VISA*, pages 19–28, 2009.

[27] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM CCR*, 33(2):65–81, 2003.

[28] G. Schaffrath, S. Schmid, and A. Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2012.

[29] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: Proposal and initial prototype. In *Proc. ACM SIGCOMM VISA Workshop*, pages 63–72, Barcelona, Spain, August 2009. ACM.

[30] K. Yi and Q. Zhang. Multi-dimensional online tracking. In *Proc. SODA*, pages 1098–1107, 2009.

[31] Y. Zhu and M. H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proc. IEEE INFOCOM*, volume 2, pages 1–12, 2006.

Dr. Marcin Bienkowski is currently an assistant professor in the Computer Science and Mathematics department at the University of Wroclaw, Poland. He received his Ph.D. degree in 2005 from the University of Paderborn, Germany, where he worked in the Algorithms and the Complexity Theory group. His research interests focus on online and approximation algorithms.

Prof. Anja Feldmann (Ph.D.) is a Professor at the Deutsche Telekom Laboratories. Previously she was a professor in the computer science department at the Technische Universität München, a professor in the computer science department at University of Saarbrücken in Germany, and a member of the IP Network Measurement and Performance Department at AT&T Labs Research. Her research interest is network performance debugging: starting from data traffic measurements over traffic characterization this leads to judging the performance implications of what we have learned. In 2011, Anja Feldmann was awarded the Gottfried-Wilhelm-Leibniz-Preis and the Berliner Wissenschaftspreis.

Johannes Grassler studies applied computer science at Beuth-Hochschule für Technik Berlin. He is a student worker with FG INET since February 2011. Before that, he worked for several years at Google and Kabel Deutschland. His professional interests include Unix systems programming and network virtualization.

Dr. Gregor Schaffrath studied computer science at University of Saarland (minor: Physics). He worked for several years at IFI Zurich, before becoming a PhD student at FG INET (T-Labs). His research interests include network virtualization and network-based intrusion detection.

Dr. Stefan Schmid studied computer science at ETH Zurich (minor: micro/macro economics, internship: CERN) and received his PhD degree from the Distributed Computing Group (Prof. Roger Wattenhofer), also at ETH Zurich. Subsequently, he worked with Prof. Christian Scheideler at the Chair for Efficient Algorithms at the Technical University of Munich (TUM) and at the Chair for Theory of Distributed Systems at Uni Paderborn. He is now a senior research scientist at Telekom Innovation Laboratories Berlin. Stefan Schmid is interested in distributed systems, and especially in the design of robust and dynamic networks.