Online Strategies for Intra and Inter Provider Service Migration in Virtual Networks

Dushyant Arora
T-Labs / TU Berlin
Berlin, Germany
dushyant@
net.t-labs.tu-berlin.de

Marcin Bienkowski
Institute of Computer Science
University of Wrocław, Poland
mbi@
ii.uni.wroc.pl

Anja Feldmann
T-Labs / TU Berlin
Berlin, Germany
anja@
net.t-labs.tu-berlin.de

Gregor Schaffrath
T-Labs / TU Berlin
Berlin, Germany
grsch@
net.t-labs.tu-berlin.de

Stefan Schmid
T-Labs / TU Berlin
Berlin, Germany
stefan@
net.t-labs.tu-berlin.de

ABSTRACT

Network virtualization allows one to build dynamic distributed systems in which resources can be dynamically allocated at locations where they are most useful. In order to fully exploit the benefits of this new technology, protocols need to be devised which react efficiently to changes in the demand. This paper argues that the field of online algorithms and competitive analysis provides useful tools to deal with and reason about the uncertainty in the request dynamics, and to design algorithms with provable performance guarantees.

As a case study, we describe a system (e.g., a gaming application) where network virtualization is used to support thin client applications for mobile devices to improve their Quality-of-Service (QoS). By decoupling the service from the underlying resource infrastructure, it can be migrated closer to the current client locations while taking into account migration cost. This paper identifies the major cost factors in such a system, and formalizes the corresponding optimization problem. Both randomized and deterministic, gravity center based online algorithms are presented which achieve a good tradeoff between improved QoS and migration cost in the worst-case, both for service migration within an infrastructure provider as well as for networks supporting cross-provider migration. We report on our simulation results and also present an explicit construction of an optimal offline algorithm which can be used, e.g., to evaluate the competitive ratio empirically.

1. INTRODUCTION

The Internet today suffers from its own success: although the Internet developed tremendously in size and speed, innovation is constrained to lower layers (e.g., to new link technologies) or to new applications "on the edge". Typical manifestations of this "ossification" include the absence of inter domain multicast support or Quality-of-Service guarantees, and the difficulties to introduce IPv6 in the public Internet. Due to its size, changing the Internet is difficult, and despite their attractive properties, clean-slate designs are problematic.

One attractive solution to enable innovation in the Internet is *network virtualization*. The concept of virtualization promises an abstraction of heterogeneous resources and provides a more efficient resource usage while ensuring isolation. This design principle has been successfully employed for a long time not only to manage the various resources of a single computer, such as memory or CPU, but today entire machines are virtualized ("node virtualization"): for example, the architecture of cloud computing systems is often fully virtualized, and renting physical machines is uncommon; rather, customers are provided with virtual machines that may share resources and that can be migrated to locations where the allocation is efficient.

Network virtualization [11] goes one step further and virtualizes not only nodes but also links (e.g., through new technologies such as OpenFlow). To the user, the virtual network appears as a physical network. However, multiple virtual networks may cohabit the same underlying network, sharing its physical links and routers. The decoupling of virtual networks from physical constraints facilitates a resource efficient embedding of the virtual networks (VNets), and may also allow for migration (as long as the specification of the virtual network is not violated).

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 the migration comes at a certain cost, good strategies are required to decide on when and where to move services. This paper revolves around this question and focuses on scenarios where the dynamics of the user demand is hard to predict.

Concretely, this paper studies a mobile thin client application (such as a *game server* [17]) that is supported by network virtualization technology. We assume that the distribution of thin clients and therefore the request pattern changes over time, e.g., due to timezone effects: for instance, at 1 a.m. GMT many requests may originate in Asian countries, then more and more requests come from European users and later from the United States. In this setting it can be beneficial to migrate (or *re-embed*) the service closer to the users, e.g., to minimize access delays for the users and to

IPTComm'11, August 1–2, 2011, Chicago, Illinois, USA.

minimize network costs for the providers [16]. Network virtualization allows us to realize such networks.

While moving services close to clients can reduce latency, migration comes at a cost: the bulk-data transfer imposes load on the network and may cause a service disruption. In particular, the cost of migration depends on the available bandwidth in the substrate network [7]. Moreover, if VNets are provisioned across administrative domains belonging to multiple infrastructure providers, (inter provider) migration entails certain transit (or *roaming*) costs. To gain insights into this tradeoff, we identify the main costs involved in this system. Intuitively, the benefits from virtualization are higher the lower the migration cost is relative to the latency penalty. A predictable access pattern may ease migration. However, in practice, user arrival patterns can be chaotic, and thus we explicitly incorporate uncertainty about future arrivals.

The classic formal tool to study algorithms that deal with inputs (or more specifically: request accesses) that arrive in an online fashion and cannot be predicted is the *competitive analysis framework*. In competitive analysis, the performance of a so-called online algorithm is compared to an optimal offline algorithm that has complete knowledge of the input *in advance*. In effect, the competitive analysis is a *worst case performance analysis* that does not rely on any statistical assumptions or prediction models. We apply this framework to network virtualization and propose—for a simplified model where, e.g., the main access cost is delay to the server and the main migration cost is the available bandwidth between migration source and destination—a competitive migration algorithm whose performance is close to the one of the optimal offline algorithm.

1.1 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] and promises to overcome the "ossification" of the Internet [12]. Virtual networks can be realized on different layers and can be supported by new technologies such as *VLANs* and *OpenFlow* [22] for traffic shaping and ensuring QoS. For a more detailed survey on the subject, please refer to [11]. We are developing our own prototype architecture, for which we also implemented a demonstrator for the migration of a media streaming server. More information on our project can be found in [29].

Embedding. A major challenge in network virtualization is the *embedding* [23] 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 [20], for an online and competitive algorithm see [13], for an embedding with only bandwidth constraints see [14], for heuristic approaches without admission control see [34], or for a simulated annealing approach see [26]. Since the general embedding problem is computationally hard, Yu et al. [21] advocate to rethink the design of the substrate network to simplify the embedding; for instance, they allow to split a virtual link over multiple paths and perform periodic path migrations. Lischka and Karl [19] 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 [9] with the help of a mixed integer program (a more general formulation which also includes migration aspects can be found in [28]). Finally, several challenges of embeddings in wireless networks have been identified by Park and Kim [24].

In contrast to the approaches discussed above we, in this paper, tackle the question of how to dynamically embed or migrate virtual servers [25] 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. [16] 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.

Migration and Online Algorithms. The possibility to migrate services without service interruption and even live has interesting implications and poses new challenges. For instance, see [8] for a recent attempt to model tradeoffs (in the cloud). To the best of our knowledge, [3] and [1] (for online server migration), and [13] (for online virtual network embeddings) are the only works to study network virtualization 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., [5, 6]) 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: this algorithm uses a (well separated) tree approximation for the general metric space (in a preprocessing step) and subsequently solves the problem on this distorted space; unfortunately, both algorithmic parts are rather complex.

In the field of *facility location*, researchers aim at computing optimal facility locations that minimize building costs and access costs (see, e.g., [15] for an online algorithm). In [18], Laoutaris et al. propose a heuristic algorithm for a variant of a facility location problem which allows for facility migration; this algorithm uses neighborhood-limited topology and demand information to compute optimal facility locations in a distributed manner. In contrast to our work, the setting is different and migration cost is measured in terms of hop count. There is no performance guarantee. In the field of *k-server problems* (e.g., [5]), an online algorithm must control the movement of a set of k servers, represented as points in a metric space, and handle requests that are also in the form of points in the space. As each request arrives, the algorithm must determine which server to move to the requested point. The goal of the algorithm is to reduce the total distance that all servers traverse. In contrast, in our model it is possible to access the server remotely, that is, there is no need for the server to move to the request's position. The *page migration problem* (e.g., [2]) 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.

There is an intriguing relationship between server migration and *online function tracking* [4, 32]. In online function tracking, an entity Alice needs to keep an entity Bob (approximately) informed about a dynamically changing function, without sending too many updates. The online function tracking problem can be transformed into a chain network where the function values are represented by

the nodes on the chain, and a sequence of value changes corresponds to a request pattern on the chain. In particular, it follows from [4] that already for some very simple linear substrate networks of size $n = \Theta(\beta)$, where β is the migration cost, no deterministic or randomized online algorithm can achieve a competitive ratio smaller than $\Omega(\log n / \log \log n)$.

Own Work and Multi-Provider Aspects. A preliminary version of this paper was presented at the VISA workshop [3] (a discussion of multiple server scenarios appeared at the Hot-ICE workshop [1]). We extend the results in [3] in several respects: First, we present a deterministic alternative to the randomized migration strategy derived in [3]; this deterministic online algorithm is based on gravity centers and achieves a better competitive ratio against adaptive adversaries. Second, we extend our model to settings with multiple infrastructure providers, and describe competitive intra and inter provider migration strategies. Provisioning virtual network services across multiple providers is an interesting topic which is hardly explored in literature so far; notable exceptions are *PolyVINE* [10], a distributed coordination protocol to perform cross-provider embeddings, or *V-Mart* [33] that describes an auction framework for task partitioning. In the context of clouds, there exists interesting literature as well, e.g., *Reservoir* [27] explores the notion of a federated cloud in which providers lease excess capacity to others in need of temporary additional resources, similarly to electrical power grids. Finally, in contrast to [3], we conducted extensive experiments to complement our formal analysis.

1.2 Contributions and Organization

This paper studies a mobile network virtualization architecture where thin clients on mobile devices access a service that can be migrated closer to the access points to reduce user latency. We identify the main costs in this system (Section 2) and introduce an optimization problem accordingly. Both for networks with a single provider (Section 3) as well as for networks with multiple providers (Section 4), online migration strategies are presented that are provably competitive to an optimal offline algorithm, i.e., that achieve a good performance even in the worst-case. This paper also describes an optimal offline algorithm which is useful for finding optimal strategies at hindsight, for dealing with regular and periodic request patterns, and for computing the competitive ratios of online algorithms (Section 5). We report on our experiments in different scenarios in Section 6. In Section 7, the paper concludes.

Note that an extended version of this article appears as ArXiv Technical Report 1103.096.

2. ARCHITECTURE

Our work is motivated by the virtualization architecture proposed in [29] for which we are in the process of developing a prototype implementation. The main roles of this architecture related to this work are: The *(Physical) Infrastructure Provider (PIP)*, which owns and manages an underlaying physical infrastructure called "substrate" (we will treat the terms *infrastructure provider* and *substrate provider* as synonyms); the *Virtual Network Provider (VNP)*, which is responsible for assembling virtual resources from one or multiple PIPs into a virtual topology; the *Virtual Network Operator (VNO)*, which is responsible for the installation and operation of a VNet over the virtual topology provided by the VNP; and the *Service Provider (SP)*, which offers application, data and content services to end-users.

We assume that a service provider is offering a service to mobile clients which can benefit from the flexibility of network and service virtualization. The goal of the service provider is to minimize the round-trip-time of its service users to the servers, by triggering migrations depending, e.g., on (latency) measurements. Concretely, VNP and/or PIPs will react on the SP-side changes of the requirements on the paths between server and access points, and re-embed the servers accordingly.

In the remainder of this paper, if not stated otherwise, the term *provider* will refer to the PIP role in the above architecture. In particular, we will study multi provider scenarios where the service provider may decide to migrate an application across PIP boundaries.

2.1 General Cost Model

Formally, we consider a substrate network $G = (V, E)$ managed by one or multiple substrate providers (PIP). Each substrate node $v \in V$ has certain properties and features associated with it (e.g., in terms of operating system or CPU power); in particular, we assume that it has a computational capacity $c(v)$. Similarly, each link $e = (u, v) \in E$, with $u, v \in V$, has certain properties, e.g., it is characterized by a bandwidth capacity $\omega(e)$, and it offers the latency $\lambda(e)$.

In addition to the substrate network, there is a set T of external machines (the mobile thin clients or simply *terminals*) that access G by issuing requests to virtualized services hosted on a set of virtual servers by G. There is a set of services $\mathcal{S} = \{S_1, S_2, ...\}$ where each service S_i can be offered by multiple servers $s \in S_i$. In the technical part of the paper we will focus on a single-service, single-server scenario only. Each server s has a certain resource or capacity requirement $r(s)$ that needs to be allocated to s on the substrate node where it is hosted.

In order for the machines in T to access the services $\mathcal{S}$, a fixed subset of nodes $A \subseteq V$ serve as *Access Points* where machines in T can connect to G. Due to the movement of machines in T, the access points can change frequently, which may trigger the migration algorithm. We define σ_t to be the multi-set of requests at time t where each element is a tuple $(a \in A, S \in \mathcal{S})$ specifying the access point and the requested service S. (For ease of notation, when clear from the context, we will sometimes simply write $v \in \sigma_t$ to denote the multi-set of access points used by the different requests.) Our main objective is to shed light onto the trade-off between the access costs Cost_{acc} of the mobile clients to the current service locations and the server migration cost Cost_{mig}: while moving the servers closer to the requester may reduce the access costs and hence improve the quality of service, it also entails the overhead of migration.

We can identify the following main parameters which influence the access and migration costs. A major share of Cost_{acc} is due to the request latency, i.e., the sum of the requests' latencies to the corresponding servers. Observe that the routing of the requests occurs along the shortest paths (w.r.t. latency) on the substrate network. In addition, the access cost depends on the server load, that is, the access cost depends on the capacity $c(v)$ of the hosting node v and the resource demands $r(s)$ of the servers s hosted by v. The correlation between load and delay can be captured by different functions, and is not studied further here. In this paper, we assume that requests are relatively small, and hence, we do not explicitly model bandwidth constraints in Cost_{acc}. In conclusion, at time t and for some function f,

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

In contrast to the requests, which are rather light-weight, the server state is typically large, and hence the traffic volume of migration cannot be neglected. The main cost of migration are service

outage periods and the migration itself. The migration cost Cost_{mig} of a virtual server $s \in S$, or the outage period, hence depends to a large extent on the available bandwidth $\omega(p)$ on the migration path $p : src \rightsquigarrow dst$ (along the substrate network) between migration source node src and destination node dst, and the size $\text{size}(s)$ of the application s to be migrated. Another major cost factor is the *transit costs*, namely the number k of PIPs on the path. In summary:

$$\text{Cost}_{\text{mig}}(t) = \sum_{s \in S} g(\omega(p), k, \text{size}(s))$$

for some function g, where the migration cost is zero if $src = dst$.

Our model so far lacks one additional ingredient: *request dynamics* (e.g., due to time-zone effects or due to user mobility). One approach would be to assume arbitrary request sets σ_t, where σ_t is completely independent of σ_{t-1}. However, for certain applications it may be more realistic to assume that the mobile nodes move "slowly" between the access points. Note that while users typically travel between different cities or countries at a limited speed, these geographical movements may not translate to the topology of the substrate network. Thus, rather than modeling the users to travel along the links of G, we consider on/off models where a user appears at some access point $a_1 \in A$ at time t, remains there for a certain period Δt, before moving to another arbitrary node $a_2 \in A$ at time $t + \Delta t$.

One may assume that Δt is exponentially distributed. However, in our formal analysis we assume a worst-case perspective and consider arbitrary distributions for Δt. Often, it is reasonable to assume some form of correlation between the individual terminal movements. For example, in an urban area, workers commute downtown in the morning and return to suburbs in the evening. Or in a planetary-scale substrate network, time zone aspects have to be taken into account in the sense that during a day, first many requests will originate from Asian countries, followed by an active period in Europe and finally America. However, as it is rather hard to describe and characterize such movement accurately we, in the formal part of this paper, perform a worst case analysis (w.r.t. latency) that does not use any statistical assumptions.

To what extent the system can benefit from virtual network support and migration depends on several factors, e.g., how frequently the thin clients change the access points. 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 paper.

2.2 Competitive Analysis

As already discussed, competitive analysis asks the question: How well does the system perform compared to an optimal offline strategy which has complete knowledge of the entire request sequence in advance? In the following, we present an online migration strategy that is "competitive" to any other online or offline solution for virtual network supported server migration. In order to focus on the main properties and trade-offs involved in the virtualization support of thin clients, we assume a simplified online framework for our formal analysis. We consider a synchronized setting where time proceeds in time slots (or *rounds*).[1] In each round t, a set of σ_t terminal requests arrive in a worst-case and online fashion at an arbitrary set of access nodes A.

[1]Note that while this assumption simplifies the analysis, it is not critical for our results.

Thus the embedding problem is equivalent to the following synchronous game, where an online algorithm ALG has to decide on the migration strategy in each round t, without knowing about the future access requests. In each round $t \geq 0$:

1. The requests σ_t arrive at some access nodes A.
2. The online algorithm ALG decides where in G to migrate the servers S. If positions are changed, it pays migration costs $\text{Cost}_{\text{mig}}(t)$.
3. The online algorithm ALG pays the requests' access costs $\text{Cost}_{\text{acc}}(t)$ to the corresponding servers (e.g, hop distance).

Note, that we allow ALG to migrate the virtual servers for all the requests of the current time slot t. However, as we assume that a request is much cheaper than a migration, and if there are not too many requests arriving concurrently, our results also apply to scenarios where the last two steps are reordered.

We aim at devising competitive algorithms ALG that minimize the *competitive ratio* ρ: Let $\text{ALG}(\sigma)$ be the total migration and access costs incurred by ALG under a request arrival sequence σ (a sequence of access points), that is,

$$\text{ALG}(\sigma) = \sum_t \text{Cost}_{\text{acc}}(t) + \text{Cost}_{\text{mig}}(t).$$

Let $\text{OPT}(\sigma)$ be the optimal cost of an offline algorithm OPT for the given σ, that is, OPT has a complete knowledge of σ and can hence optimize the server locations "offline". ρ is the ratio of the costs of ALG and OPT. Thus, our objective is to minimize:

$$\rho = \max_{\sigma} \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)}$$

In case of online algorithms that use randomization, we consider the expected costs against an oblivious adversary without access to the outcome of the random coin flips of the algorithm.

For our analysis, we make the following simplification. The access cost $\text{Cost}_{\text{acc}}(t)$ is given by the latency on the shortest path from the request to the service. Moreover, the migration cost $\text{Cost}_{\text{mig}}(t)$ is given by the bandwidth constraint of the smallest edge capacity on the migration path, plus the number of PIPs traversed times π. Let $\text{Cost}_{\text{mig}}(u, v)$ denote the migration cost on a path from u to v. (The path will be clear from the context.) Thus, $\text{Cost}_{\text{mig}}(u, v) = \max_e \text{size}(s)/\omega(e) + k \cdot \pi$ where $\text{size}(s)$ is the size of the migrated server s, e is a link on the migration path from u to v, k is the number of PIPs traversed and π is the cost of migrating across a PIP boundary. Observe that given a migration path, the cost is different in case the migration occurs once along the entire path compared to the case where it occurs in two steps at different times.

3. INTRA PROVIDER MIGRATION

This section presents two online protocols for single server migration within a single provider (PIP). The main idea of both algorithms is to divide time into *epochs*: As we will see, also an optimal offline algorithm will have certain costs in such an epoch, which allows us to compare its performance to the online algorithms. While algorithm MIX uses randomization to identify good locations to serve the current requests, algorithm CEN deterministically migrates to the gravity centers of the demand. Before describing these two algorithms in detail, we briefly discuss static strategies without migration.

3.1 The STAT Algorithm

In order to compare the benefits of migration to a static scenario, we derive the competitive ratio of fixed strategies (see also [3]).

LEMMA 3.1. *A system without migration yields a competitive ratio of*

$$\rho \in \Theta(\mathit{Diam}(G)),$$

where $\mathit{Diam}(G)$ is the network diameter of substrate network G. [3]

In a fixed scenario, the best static algorithm STAT hosts s is in the network center, i.e., the location which minimizes the worst-case distance traveled by the requests, namely at node u for which $u := \arg\min_{v \in V} \max_{w \in V} \text{Cost}_{\text{acc}}(v, w)$.

Note, since there is no migration in the fixed scenario, the competitive ratio does not depend on any bandwidth constraints (i.e., on link weights). This means that in networks with highly heterogeneous links or with links whose capacity changes quickly over time, a static solution without migration may be good.

3.2 The MIX Algorithm

We now describe an online migration algorithm MIX (see also [3]). The basic idea of MIX is to strike a balance between the request latency cost $\text{Cost}_{\text{acc}}^{\text{MIX}}$ and the migration cost $\text{Cost}_{\text{mig}}^{\text{MIX}}$ it incurs, and to continuously move closer to a possible optimal position. The intuition is that after a small number of migrations only, either MIX is at the optimal position, or an optimal offline algorithm OPT must have migrated as well during this time period. Either way, OPT cannot incur much smaller costs than MIX. In other words, by using MIX for moving to good locations in the network, a possible offline algorithm that migrates less frequently cannot have much lower access costs than MIX; on the other hand, an offline strategy with frequent migrations will have similar costs to Cost_{mig}.

Let us first consider a scenario with constant bandwidth capacities, i.e., $\omega(e) = \omega \ \ \forall e \in E$ and let $\beta = \text{size}(s)/\omega$ be the corresponding migration cost.

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 denote this counter by $\text{C}(v)$. MIX keeps the server at a single node w till $\text{C}(w)$ reaches β. In this case, MIX migrates the server to a node u chosen uniformly at random among nodes with the property $\text{C}(u) < \beta$. 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 and the counters $\text{C}(v)$ are reset to zero.

LEMMA 3.2. MIX *is $O(\log n)$-competitive in networks with constant bandwidth. [3]*

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 β, and thus in either case $\text{OPT}(\mathcal{E}) \geq \beta$.

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 β. In [3], we 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 number of phases is then $H_n + 1$, and hence $\text{MIX}(\mathcal{E}) \leq \beta \cdot H_n + \beta \cdot (H_n + 1) = \beta \cdot O(\log n)$. This yields the competitiveness of MIX. □

Note that the 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.

For networks with general bandwidths, MIX can be adopted in such a way that it migrates when the counter of the current location v reaches $\text{size}(s)/\min_e \omega(e)$, that is, when $\text{C}(v) \geq \text{size}(s)/\min_e \omega(e)$. Thus, the cost of the optimal algorithm in each epoch is at least $\text{size}(s)/\max_e \omega(e)$, while the cost of MIX is at most $\text{size}(s)/\min_e \omega(e)$. Therefore, by the same arguments as in the proof of Lemma 3.2, we immediately obtain the following result.

THEOREM 3.3. MIX *is $O(\mu \cdot \log n)$-competitive in general networks, where $\mu = \max_{e,e' \in E} \omega(e)/\omega(e')$.*

3.3 The CEN Algorithm

While for the analysis of MIX, we assumed an oblivious adversary which cannot be adaptive with respect to the random choices made by the online algorithm, we now focus on deterministic algorithms CEN. As we will see, a logarithmic competitive ratio can also be achieved. Again, we will first assume $\omega(e) = \omega \ \ \forall e \in E$ and $\beta = \text{size}(s)/\omega$.

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. Henceforth, we will call all nodes v for which at time t, $\text{C}(v) < \beta/40$, *active nodes* at time t. Assume that algorithm CEN is currently at some node v. CEN remains at this node until it accumulated there access costs of β. Then, a new phase starts, and CEN computes the *gravity center* w, i.e., the "center" of the currently active nodes. Formally, let d denote the shortest path metric (w.r.t. access costs) on the network G. The gravity center of a subset $V' \subseteq V$ of nodes is defined as the (not necessarily unique) node $\mathcal{G}(V') = \arg\min_{v \in V'} \sum_{u \in V'} d(u, v)$. (Ties are broken arbitrarily.) 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 fact that a request always increases the counter of several nodes besides the gravity center (namely: a constant fraction) by at least a certain value (again, a constant fraction) as well.

LEMMA 3.4. *Let $\lambda_1 = 1/5$ and $\lambda_2 = 1/4$. Fix any active set V'. Let r be an arbitrary requesting node (at some step). Assume the counter at the gravity center $\mathcal{G}(V')$ increased by F because of this request. Then there are at least $\lambda_2 \cdot |V'|$ nodes from V' whose counters increased at least by $\lambda_1 \cdot F$.*

PROOF. Assume the contrary. It means that there are at least $(1-\lambda_2) \cdot |V'|$ nodes from V' whose counter increase is smaller than $\lambda_1 \cdot F$. Denote this set by V''. We know that the distance between the request and the center is $d(\mathcal{G}(V'), r) = F$, and $\forall u \in V''$, $d(u, r) < \lambda_1 \cdot F$. Therefore, $\forall u, v \in V''$, $d(u, v) < 2\lambda_1 \cdot F$: the diameter of the set V'' is relatively small.

Now let ξ be any node of V''. We show that ξ would be a better candidate for the gravity center than $\mathcal{G}(V')$ is. Using triangle

inequalities, we obtain

$$\begin{aligned}\sum_{u\in V'} d(\mathcal{G}(V'),u) &= \sum_{u\in V''} d(\mathcal{G}(V'),u) + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\\ &\geq \sum_{u\in V''} \big[d(\mathcal{G}(V'),r) - d(u,r)\big]\\ &\quad + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\\ &> (1-\lambda_1)\cdot|V''|\cdot F + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\\ &= \frac{4}{5}\cdot|V''|\cdot F + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\ ,\end{aligned}$$

because $d(\mathcal{G}(V'),r) = F$ and $d(u,r) \leq \lambda_1 \cdot F$, and by substituting $\lambda_1 = 1/5$. On the other hand, note that $|V' \setminus V''| \leq |V'|/4 \leq |V''|/3$ and

$$\begin{aligned}\sum_{u\in V'} d(\xi,u) &= \sum_{u\in V''} d(\xi,u) + \sum_{u\in V'\setminus V''} d(\xi,u)\\ &< 2\lambda_1\cdot|V''|\cdot F\\ &\quad + \sum_{u\in V'\setminus V''} \big[d(\xi,r) + d(r,\mathcal{G}(V')) + d(\mathcal{G}(V'),u)\big]\\ &< 2\lambda_1\cdot|V''|\cdot F + |V'\setminus V''|\cdot(1+\lambda_1)\cdot F\\ &\quad + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\\ &\leq \frac{4}{5}\cdot|V''|\cdot F + \sum_{u\in V'\setminus V''} d(\mathcal{G}(V'),u)\end{aligned}$$

because $d(\xi,r) < \lambda_1 \cdot F$, $d(r,\mathcal{G}(V')) = F$, and by substituting the value of $\lambda_1 = 1/5$. This contradicts that $\mathcal{G}(V')$ is the gravity center of V'. □

From Lemma 3.4 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 3.5. *Fix any threshold τ. When the counter at the gravity center $\mathcal{G}(V')$ exceeds τ, then there exists $V'' \subseteq V'$, $|V''| \geq \frac{1}{8}\cdot|V'|$, such that for all $v \in V''$, the counter at v is at least $\tau/40$.*

PROOF. Assume the contrary. This means that there exists $V'' \subseteq V'$, $|V''| > \frac{7}{8}\cdot|V'|$, such that for all $v \in V''$, the counter at v is smaller than $\tau/40$. Hence $\sum_{v\in V''} \mathrm{C}(v) < |V''|\cdot\tau/40 \leq |V'|\cdot\tau/40$. On the other hand, by Lemma 3.4, each time the counter $\mathrm{C}(\mathcal{G}(V'))$ increases by F, at least $1/4\cdot|V'|$ counters from set V' (and hence at least $1/8\cdot|V'|$ counters from set V'', since $|V'\setminus V''| \leq \frac{|V'|}{8}$) increase by $F/5$. Hence, in this case, the sum of counters from V'' increases at least by $1/40\cdot|V'|\cdot F$. Therefore, when $\mathrm{C}(\mathcal{G}(V')) \geq \tau$, $\sum_{v\in V''} \mathrm{C}(v) \geq |V'|\cdot\tau/40$, which is a contradiction. □

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

THEOREM 3.6. CEN *is $O(\log n)$-competitive.*

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 also in the order of $\Omega(\beta)$. Regarding CEN, we know that in each phase, access costs are at most β, and it remains to study the number of phases per epoch. By Lemma 3.5, we know that in each phase, the number of active nodes is reduced by a factor at least $1/8$. Therefore, there are at most $O(\log n)$ many phases per epoch, and the claim follows. □

Note that we did not try to optimize the constants in this proof, and in practice (and in our simulations), alternative thresholds can be applied yielding better (but qualitatively equivalent) results.

Again, for networks with general bandwidths, CEN can be adopted in such a way that it migrates when the counter of the current location v reaches $\mathrm{size}(s)/\min_e \omega(e)$, that is, when $\mathrm{C}(v) \geq \mathrm{size}(s)/\min_e \omega(e)$. By the same arguments as above, this adds a factor $\max_{e,e'\in E} \omega(e)/\omega(e')$ to the competitive ratio.

3.4 Remarks

Recall that the adversarial model for MIX and CEN is different, and hence, 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 the question which of the two strategies is more efficient depends on the scenario.

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. This allows us to generalize our analysis to scenarios where the access latency, in addition to the sum of the link latencies, depends also on the capacity of the hosting node: We simply need to take the capacities into account when increasing the counters. In case of CEN, the access costs must fulfill the triangle inequality.

4. INTER PROVIDER MIGRATION

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 extend our model to multiple provider scenarios. In particular, we assume that migrating a server across a PIP boundary 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.

In order to study the benefits of migration, we again consider a scenario without migration (algorithm STAT). Of course, Lemma 3.1 still applies: in a fixed scenario, the best location for hosting s is in the network center, i.e., the location which minimizes the distance traveled by the requests.

In the following, we present how the randomized algorithm MIX and the deterministic algorithm CEN can be extended to multi-PIP scenarios. We consider k PIPs, migration inside a PIP costs β, access costs are the number of hops, and migrating across providers costs π per crossed PIP boundary. We will concentrate on the more realistic case where $\pi \geq \beta$.[2]

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 of PIPs to be traversed on a shortest migration path, by Δ.

[2]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.

Algorithm MIX$_k$ generalizes MIX by moving the server to one of the PIPs having lower costs.

The algorithm MIX$_k$ divides time into three types of *epochs*: *huge epochs* which consist of one or several *large epochs* which in turn consist of $\lceil \pi/\beta \rceil$ *small epochs*. For each node u, we use two counters $\mathrm{C}(u)$ and $\mathrm{C}_L(u)$ to count the access cost during a small and a large epoch, respectively. At the beginning of a small epoch, all nodes are *active*; similarly, at the beginning of a huge epoch, we say that all PIPs are *active*. During a small epoch, the server is migrated within a single PIP only, until there is no node u left with access costs smaller than β: MIX$_k$ monitors, for each node u, the cost of serving all requests from this small epoch by a server kept at u; MIX$_k$ keeps the server at a single node u till $\mathrm{C}(u)$ reaches β. When this happens, MIX$_k$ migrates the server to a node v chosen uniformly at random among nodes of the current PIP with the property $\mathrm{C}(v) < \beta$. If there is no such node, MIX$_k$ does not migrate the server, and the small epoch ends in that round; the next epoch starts in the next round and the counters $\mathrm{C}(u)$ are reset to zero.

After $\lceil \pi/\beta \rceil$ small epochs a large epoch ends. Then MIX$_k$ determines the set of PIPs that contain at least one node v for which $\mathrm{C}_L(v) < \pi$; all other PIPs become inactive for the remainder of the current huge epoch. If there are active PIPs left, MIX$_k$ chooses an active PIP uniformly at random and migrates to an arbitrary node of that PIP; otherwise the server stays where it is, and a new huge epoch begins.

We can derive the following competitive ratio on MIX$_k$'s performance.

THEOREM 4.1. *MIX$_k$ is $O(\log k \cdot (\log n_1 + \Delta))$-competitive in networks with constant bandwidth and k PIPs, where n_1 is the size of the largest PIP, and Δ is the "diameter of the PIP graph".*

PROOF. From Lemma 3.2, we know that during a small epoch, MIX$_k$ accumulates a cost of at most $O(\beta \log n_1)$: There is at most a logarithmic number of migrations, and the access costs per phase is at most β. Recall that a large epoch consists of at most $\lceil \pi/\beta \rceil$ many small epochs, and subsequently, a remaining active PIP is chosen uniformly at random. Thus, similarly to Lemma 3.2, it holds that there are at most $O(\log k)$ many large epochs, yielding a total access cost of $O(\log k \cdot \log n_1 \cdot \pi/\beta \cdot \beta) = O(\pi \log k \cdot \log n_1)$. The migration costs within PIPs are of the same order. The transit costs to move the server between PIPs amounts to at most $O(\Delta \pi \log k)$. Thus, the overall cost of MIX$_k$ per huge epoch is in the order of $O(\pi \log k \cdot (\log n_1 + \Delta))$. On the other hand, an optimal offline algorithm must have had costs of at least π as well during this huge epoch: if the optimal algorithm migrates between PIPs, the claim follows trivially. Otherwise, the optimal offline algorithm is located at a single PIP during the entire huge epoch; by the construction of MIX$_k$, there must exist a large epoch in which the optimal offline algorithm incurred a cost of at least $\Omega(\pi)$: per small epoch the (access or migration) costs are at least β, and there are $\lceil \pi/\beta \rceil$ many small epochs in a large epoch. The claim follows. □

Note that the proof of Theorem 4.1 is overly pessimistic, as it assumes several large migration distances that the optimal offline algorithm can avoid. We believe that MIX$_k$ performs better, also in the worst-case, a conjecture that is also manifested by our experiments.

A similar extension also works for the deterministic variant.

The algorithm CEN$_k$ divides time into three types of *epochs*: a *huge epoch* consists of multiple *large epochs*, and a large epoch consists of $40\lceil \pi/\beta \rceil$ *small epochs*. 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. CEN$_k$ then monitors, for each node u, the cost of serving all requests from this small epoch by a server kept at u. CEN$_k$ leaves the server at a single node u till $\mathrm{C}(u)$ reaches β. In this case, CEN$_k$ 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 $\mathrm{C}(w) < \beta/40$. 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 $40\lceil \pi/\beta \rceil$ small epochs, a new large epoch starts, and all nodes u in the network with $\mathrm{C}_L(u) \geq \pi/40$ become inactive with respect to the large epoch. Among all remaining active nodes of the large epoch, CEN$_k$ 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.

We can show the following result.

THEOREM 4.2. *CEN$_k$ is $O(\log n \cdot (\log n_1 + \Delta))$-competitive in networks with constant bandwidth and k PIPs, where n_1 is the size of the largest PIP, and Δ is the diameter of the PIP graph.*

PROOF. First we compute the total cost of CEN$_k$ in a huge epoch. It follows from Theorem 3.6 that a large epoch consists of $40\lceil \pi/\beta \rceil$ small epochs of $O(\beta \log n_1)$ access costs and at a logarithmic number of migrations amounting to cost $O(\beta \log n_1)$ as well, yielding a total cost per large epoch of $O(\pi \log n_1)$. Now observe that there is at most a logarithmic number of large epochs per huge epoch: CEN$_k$ guarantees that the server is not migrated to another PIP as long as there is a node left in the current PIP with access costs smaller than π; in particular, for the center of gravity u of the current large epoch PIP, $\mathrm{C}_L(u) \geq \pi$, and hence, again by Theorem 3.6, a constant fraction of nodes in the entire network must become inactive per large epoch. Summing up over the large epochs and adding the transit cost of at most $O(\Delta \pi \log n)$, the total cost is at most $O(\log n \cdot \pi(\log n_1 + \Delta))$. The cost of the optimal offline algorithm can be analyzed similarly to the proof of Theorem 4.1: If the offline algorithm migrates during a huge epoch, it has a cost of at least π; otherwise, it has either access or migration costs of at least $\beta/40$ per small epoch and hence $\Omega(\pi)$ per large epoch, and the claim follows. □

Again, we believe that the actual ratio is better, even in the worst-case, as our analysis is pessimistic.

4.1 Remarks

Note that MIX$_k$ has the attractive property that it poses minimal assumptions on the knowledge of the infrastructure topology and allow for a large autonomy on the PIP level. Typically, substrate providers are known for their secrecy on traffic matrices as well as topology information. All information needed by MIX$_k$ is the set of providers that could have served the requests of a certain time period at lower cost, e.g., the set of providers that could make "a better offer". CEN$_k$ on the other hand requires more knowledge of the topology. It assumes that gravity centers can be computed across PIP boundaries, which is unrealistic. However, while this

facilitates the formal analysis, we believe that pragmatic implementations that move the service, e.g., to the PIP which lies "at the center" of the active providers, yield good approximations and justify the validity of concept and analysis.

5. OPTIMAL OFFLINE ALGORITHMS

In the competitive analysis of our online algorithms we often argued about a hypothetical optimal offline algorithm to which we compare our costs; there was no need to find or describe the offline algorithm explicitly. However, while the decisions when and where to migrate servers typically needs 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.

This section is based on the ideas described in the VISA workshop version of this paper [3]. We present an optimal offline algorithm for our server migration problems. It turns out that offline strategies can be computed for many different scenarios, and we describe a very general algorithm here. Similarly to the online algorithms, offline strategies can be computed efficiently both for intra and inter provider migration.

It exploits the fact that migration exhibits an optimal substructure property: Given that at time t, the server is located at a given node u, then 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 essentially fills out a matrix opt[time][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, the service is located at node v_0. Thus, initially, $opt[0][u] = Cost_{\text{mig}}(v_0, u) + \left[\sum_{v \in \sigma_0} Cost_{\text{acc}}(v, u)\right]$ as the migration origin is v_0, and as a request needs to travel on the access link from the terminal to v and from there to u (w.l.o.g., we assume that the cost $Cost_{\text{acc}}$ contains the first wireless hop from terminal to substrate network).

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, in order to find the optimal cost to arrive at a configuration with server at node u at time t:

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

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

We have the following runtime result.

THEOREM 5.1. *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, which implies the claim. □

Note that OPT is not an online algorithm. Although $opt[t]$ does not depend on future requests, in order to reconstruct the optimal migration strategy at hindsight, the configuration of minimal cost after the last request is determined, and from there, the optimal path is given by recursively finding the optimal configuration at time $t-1$ which led to the optimal configuration at time t.

6. SIMULATIONS

In order to complement our formal insights and in order to study the behavior of our algorithms in different environments, we implemented a simulation framework. In the following, we report on some of our results in more detail.

6.1 Set-Up

We conducted experiments on both artificial *Erdös-Rényi* random graphs (with connection probability 1%) as well as more realistic graphs taken from the *Rocketfuel project* [30, 31] (including the corresponding latencies for the access cost). Due to space constraints, the discussion of the latter is included in the ArXiv technical report only.

If not stated otherwise, we assume that link bandwidths are chosen at random (either T1 (1.544 Mbit/s) or T2 (6.312 Mbit/s)), that the server size is 2048MB, that β equals the server size divided by the average bandwidth, and $\pi = 3\beta$.

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. However, as our online algorithms have a much lower runtime than OPT, experiments that do not rely on optimal offline results can be conducted for much more nodes. Moreover, to gain insights into the behavior of our algorithms in networks of this size, we use a threshold $\tau = 1/3$ (rather than $\tau = 1/40$) to inactivate nodes in CEN. This value is more practical and does not change the qualitative results for large networks.

As the real traffic patterns are subject to confidentiality, we consider two different simplified, artificial scenarios. Our scenarios assume that the substrate topology does not reflect the geographic situation or user pattern at all. This is conservative of course, and online migration algorithms typically perform better if requests move along the topology.

Time Zones Scenario: This scenario models an access pattern that can result from global daytime effects. We divide a day into T time periods. At each time t, $p\%$ of all requests originate from a node chosen uniformly at random from the substrate network (pessimistic assumption). The sojourn time of the requests at a given location is distributed exponentially with parameter λ as well. In addition, there is background traffic: the remaining requests originate from nodes chosen uniformly at random from all access points.

We also studied an alternative scenario, capturing traffic from commuters.

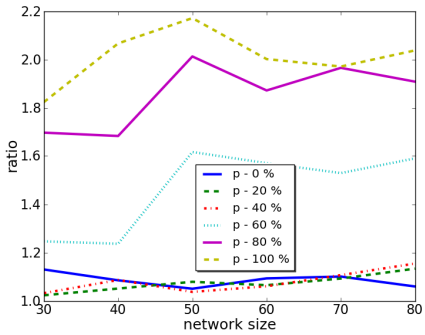

Figure 1: Competitive ratio of CEN as a function of network size and p in a time zone scenario. Results are averaged over 10 runs, we use $\lambda = 10$ and collected data over a period of 400 rounds.

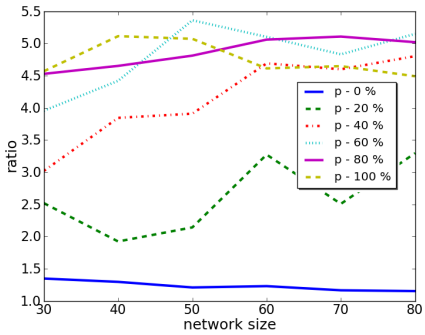

Figure 2: Same experiment as Figure 1 for MIX.

Commuter Scenario: This scenario models an access pattern that can result from commuters traveling downtown for work in the morning and returning back to the suburbs in the evening. We use a parameter T to model the *frequency* of the changes. At time $t \bmod T < T/2$, requests originate from $2^{t \bmod T}$ access points chosen uniformly at random around the center of the network. In the second half of the day, i.e., for $t \in [T/2, ..., T]$, the pattern is reversed. Then a new day starts. The commuter scenario can come in different flavors, e.g., where the total number of requests remains constant over time, or where the load is changing. We use the static load scenario in our simulations. The total number of requests per round is fixed to $2^{T/2}$. At time $t_i < T/2$, the requests originate from $p = 2^{t_i \bmod T}$ of all access points including the network center ($2^{T/2}/p$ requests per access point), until single requests originate from $2^{T/2}$ access points. Then, the same process is reversed until all $2^{T/2}$ requests originate from a single access point: the network center. We assume that the time period between t_i and t_{i+1} is distributed exponentially with parameter λ.

6.2 Intra Provider Migration

A first set of experiments studies the competitive ratio as a function of the number of nodes. Figure 1 reports on the impact of different correlations of the requests in the time zone scenario. First, we can observe that the competitive ratios are generally quite low, and more or less independent of the network size. In order to take into account that larger networks typically come with a larger requests set, we assume that the number of requests per round is one fifth of the network size. We can see that the competitive ratios of CEN are again quite low, but the optimal offline algorithm can do relatively better if p is large, which meets our intuitions.

The same results for MIX can be seen in Figure 2. Again, larger p yield higher ratios, and generally the performance is worse than the one of CEN.

Another interesting question regards how the competitive ratio depends on the dynamics, i.e., on λ. Figure 3 presents our results for MIX and CEN. Although the variance is quite high (this is typical, especially for MIX), we can see a trend that in case of very high dynamics and very low dynamics, the competitive ratio is slightly lower than for λ values in-between (λ is the mean stay duration). This can be explained by the fact that OPT can optimize relatively more in scenarios where the online migration decisions are not obvious.

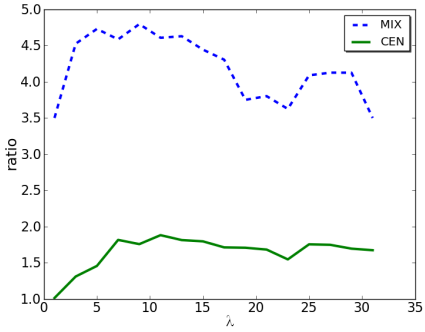

Figure 3: Competitive ratio in time zone scenario (with $p = 60\%$) as a function of λ and averaged over 10 runs and in a network of 60 nodes. We ran the experiment for 200 rounds.

We note that for the random graphs of the size used in our experiments, the diameter is relatively low and hence STAT typically has a good performance as well.[3] Figure 4 shows that both online algorithms perform well (and better than in the time-zone scenario). One takeaway from our experiments with the different scenarios is that migration is relatively more beneficial in these instances of the commuter scenario compared to the time zone instances studied.

6.3 Inter Provider Migration

We also conducted some experiments with multiple PIPs. Generally, we used the same random topologies to model a single PIP network, and connected different PIPs in a circular manner using a random connection between adjacent providers.

As in the intra provider scenario, we first report on scalability. Figure 5 plots the competitive ratio as a function of the total number of nodes per provider, given that there are three providers. The ratios are similar to the single PIP case, MIX$_k$ is slightly worse than CEN$_k$, and constant bandwidth scenarios yield lower ratios.

Figure 6 shows the effect of different correlation (p values) for CEN$_k$, and Figure 7 studies the analogous situation for MIX$_k$.

It turns out that both MIX$_k$ and CEN$_k$ are relatively robust to different values of x, the relative cost of transit compared to migration, although MIX$_k$ slightly benefits from lower migration costs, as we would expect (see Figure 8).

[3] In additional experiments, we observed that in the commuter scenario, it is generally worse than in the time zone scenario.

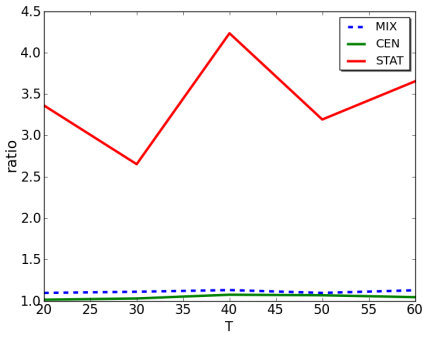

Figure 4: Commuter scenario with $T = 3$, server size 30MB $\lambda = 10$, runtime 200 rounds and averaged over 20 runs.

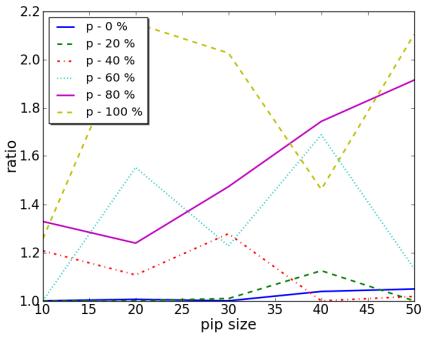

Figure 6: Time zone scenario with three PIPs, $\lambda = 5$, runtime 200 rounds: competitive ratio of $\textsc{Cen}_k$ as a function of provider size and p.

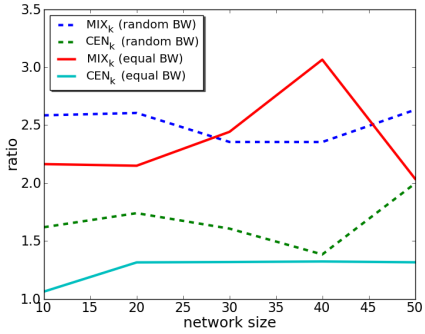

Figure 5: Competitive ratio in time zone scenario (with $p = 0\%$) with three providers, server size 30MB, $\lambda = 2$ and a runtime of 50 rounds. We average the ratio over five runs.

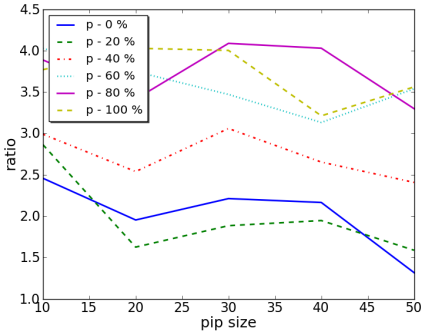

Figure 7: Like Figure 6 but for $\textsc{Mix}_k$.

7. 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, algorithms need to be designed that adapt dynamically to the current demand; 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 paper 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, both for inter and intra provider scenarios, 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 believe that the bounds on the competitive ratio for multiple PIPs are overly pessimistic and can be improved. We also plan to study (simplified versions of) our algorithms in the wild, i.e., in our prototype [29] architecture; indeed, we already implemented a demonstrator which migrates a streaming server between different locations in a German network.

Finally, we emphasize that while our formal considerations may give insights into the benefits of this new technology, e.g., in terms of improved quality of service, whether and how mobile network provider will adapt such an approach also depends on many economic factors that are not taken into account in our model.

Acknowledgments

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. We would like to thank our colleagues in these projects for many fruitful discussions; in particular: Dan Jurca (now at Huawei Technologies Duesseldorf GmbH), Wolfgang Kellerer, Ashiq Khan, Kazuyuki Kozu, and Joerg Widmer (now at Institute IMDEA Networks).

Special thanks go to Ernesto Abarca who was a great help during the prototype implementation. We also thank Johannes Grassler and Lukas Wöllner for their help with the prototype and the migration demonstrator. M. Bienkowski is supported by MNiSW grants number N N206 368839, 2010–2013 and N N206 257335, 2008–2011.

8. REFERENCES

[1] 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)*, 2011.

[2] M. Bienkowski. Migrating and replicating data in networks. *Computer Science - Research and Development*, 2011.

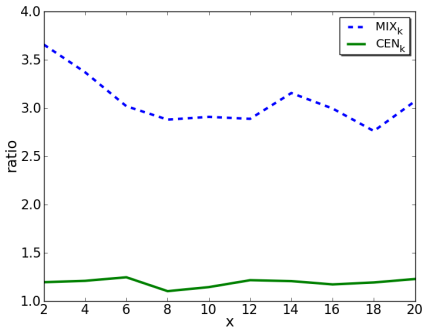

Figure 8: Ratio as a function of $x = \pi/\beta$. Time zone scenario ($p = 50\%$) with 3 PIPs, runtime 400, PIP size 20 nodes, $\lambda = 5$, averaged over 10 iterations.

[3] 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*, 2010.

[4] M. Bienkowski and S. Schmid. Online function tracking with generalized penalties. In *Proc. 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, 2010.

[5] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[6] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[7] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proc. VEE*, 2007.

[8] 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)*, 2011.

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

[10] M. Chowdhury, F. Samuel, and R. Boutaba. PolyViNE: Policy-based virtual network embedding across multiple domains. In *Proc. ACM SIGCOMM VISA*, 2010.

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

[12] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. SIGCOMM*, 2002.

[13] G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and deterministic embeddings of virtual networks. In *ArXiv Technical Report 1101.5221*, 2011.

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

[15] D. Fotakis. On the competitive ratio for online facility location. In *Proc. 30th International Conference on Automata, Languages and Programming (ICALP), also appeared in Algorithmica 50(1), pp. 1-57, 2008*, pages 637–652, 2003.

[16] 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.

[17] U. C. Kozat, Y. Gwon, and R. Jain. An overlay server system (oss) platform for multiplayer online games over mobile networks. In *Proc. Global Telecommunications Conference, 2006 (GLOBECOM)*, 2006.

[18] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros. Distributed placement of service facilities in large-scale networks. In *Proc. IEEE INFOCOM*, 2007.

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

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

[21] J. R. M. Yu, Y. Yi and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, Apr 2008.

[22] 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.

[23] B. Monien and H. Sudborough. Embedding one interconnection network in another. In *Computational Graph Theory*, 1990.

[24] 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.

[25] 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.

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

[27] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz, and G. Toffetti. ReservoirŮwhen one cloud is not enough. *IEEE Computer*, 2011.

[28] G. Schaffrath, S. Schmid, and A. Feldmann. Generalized and resource-efficient VNet embeddings with migrations. In *ArXiv Technical Report 1012.4066*, 2011.

[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*, 2009.

[30] N. Spring, R. Mahajan, and T. Andersonr. Quantifying the causes of path inflation. In *Proc. SIGCOMM*, 2003.

[31] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.

[32] K. Yi and Q. Zhang. Multi-dimensional online tracking.

[33] F. Zaheer, J. Xiao, and R. Boutaba. Multi-provider service negotiation and contracting in network virtualization. In *Proc. 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, 2010.

[34] Y. Zhu and M. H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proc. IEEE INFOCOM*, 2006.