Nomadic Service Points

Edward Bortnikov Israel Cidon Idit Keidar
Department of Electrical Engineering
The Technion, Haifa 32000
Israel
ebortnik@techunix.technion.ac.il {cidon,idish}@ee.technion.ac.il

November 19, 2005

Abstract

We consider the novel problem of dynamically assigning application sessions of mobile users or user groups to service points. Such assignments must balance the tradeoff between two conflicting goals. On the one hand, we would like to connect a user to the closest server, in order to reduce network costs and service latencies. On the other hand, we would like to minimize the number of costly session migrations, or handoffs, between service points. We tackle this problem using two approaches. First, we employ algorithmic online optimization to obtain algorithms whose worst-case performance is within a factor of the optimal. Next, we extend them with opportunistic versions that achieve excellent practical average performance and scalability. We conduct case studies of two settings where such algorithms are required: wireless mesh networks with mobile users, and wide-area groupware applications with or without mobility.

1 Introduction

Recent advances in network technology, along with the increasing demand for real-time networked applications, are bringing application service providers to deploy multiple geographically dispersed service points, or servers. This trend is expected to further expand with the explosion of new applications and the expansion of services to larger domains. In such settings, a given application session is typically associated with some server. In real-time applications, the association selection is driven by quality of service (QoS) considerations, which may depend, e.g., on the network distance of the user from the server. As many of these applications are becoming increasingly available to mobile users and dynamic user groups, the factors that dictate the server selection can vary with time. For example, due to a user's movement, a server providing optimal QoS at some point may later provide poor QoS, rendering it desirable to migrate the application session from one physical server to another. We therefore believe that many future distributed service infrastructures will employ *nomadic service points*, and will transparently manage such dynamic session assignments.

One important domain where nomadic service points can be exploited to serve mobile users is wireless mesh networks (WMNs) [1, 7, 9]. WMNs provide an increasingly popular solution for Internet access from residential areas with a limited wired infrastructure. These networks are built around multiple stationary wireless routers. Some of them, called access gateways, are wired to the Internet. A greedy routing protocol would route the traffic via the closest gateway as a mobile user travels away from its original location, causing frequent session interruptions. Intelligent nomadic service points can eliminate this problem.

Server assignment quality also has special importance in collaborative groupware applications like instant messaging, push-to-talk, and massively multiplayer online games, where the impact of a bad association can be magnified with the group's scale. The infrastructure for these applications is typically based on servers that both maintain the application state and act as forwarding proxies. Intuitively, the server should reside close to the group's centroid in order to serve the group best. In groups with a highly dynamic membership, the optimal server selection changes as users join or leave the group.

In this paper, we introduce the problem of optimizing the dynamic assignment of sessions to nomadic service points. Such a service assignment should balance the tradeoff between connecting sessions to the closest servers at all times, and minimizing the number of session migrations. We capture this tradeoff by assuming two types of service costs: a *setup* cost, incurred whenever the session is assigned to a new server, and a *hold* cost, incurred every unit of time the server is being used. The former reflects one-time expenses like signaling overhead and application state transfer, whereas the latter captures continuous expenses like buffer space, processing power, network latency, and bandwidth. We define the *nomadic server assignment* optimization problem as a sequence of server assignments that minimizes the total cost. Obviously, we are interested in the *online* version of this problem, in which the service costs are received on the fly.

We treat the problem both as a theoretical online optimization problem and as a practical system question. To the best of our knowledge, this problem was not previously addressed using either of these methodologies. We first treat the generic *nomadic service problem*, independent of any specific application domain, and then examine it more closely in two specific case studies pertaining to specific example domains.

We formally define the problem in Section 2. Then, in Section 3, we present an *offline* algorithm, `OPT`, which computes the optimal solution assuming that the costs are known in advance. This algorithm's time and space computation complexity is linear in the number of servers k and in the algorithm's duration. While this result has little practical importance, it serves as a baseline for evaluating the online algorithms described in later sections.

In Section 4, we study nomadic server assignment as an online optimization problem. We focus on the case where the setup costs do not vary over time, and are identical for all servers; the hold costs may vary in both aspects. A common metric for an online algorithms is its *competitive ratio*, which is the worst-case ratio between the cost produced by the algorithm and the optimal cost. We first prove a lower bound of k on the competitive ratio of any deterministic online assignment algorithm. We then present two simple online algorithms, `DTrack` (deficit tracker) and `CTrack` (cost tracker), parameterized by policies governing *when* transitions happen and *which* server is chosen upon a transition. `DTrack` transitions from its currently assigned server when the session accumulates "significantly more" hold cost than it would have paid had it been assigned to some other server, whereas `CTrack` simply transitions when the session accumulates "enough" hold cost at the currently assigned server. We show that when instantiated with certain policies, these algorithms achieve competitive ratios within a constant factor of the lower bound. Specifically, when using a round-robin (RR) policy to choose the next assignment, `DTrack` achieves a competitive ratio of $2k$, i.e., at most twice as bad as the lower bound, whereas `CTrack` achieves a competitive ratio of $(2 + a)k$, where a is an upper bound on the ratio between the hold and setup costs.

Although, as our lower bound shows, a worst-case cost ratio that is linear in the number of servers is inevitable in the general case, achieving such costs is hardly useful for large-scale services that employ thousands of servers world-wide. From a practical perspective, it is more interesting to examine average costs in common scenarios, and moreover, it is highly desirable for costs not to increase significantly with the number of servers. We address these practical issues in Sections 5 and 6, via empirical case studies of a WMN with mobile users and an Internet chatroom with dynamic groups, respectively. Interestingly, the competitive versions of `DTrack` and `CTrack`, which achieve the best worst-case costs, are not very promising in practice. However, *opportunistic* versions of these algorithms, which select the next assignment based on current or past offered costs (rather than in a round-robin manner), achieve excellent results. Their costs are at most 50% above the optimum in the average case in the WMN, and at most 20% above optimal in the groupware service. More importantly, this ratio, as well as the total cost, remains almost constant as the problem size scales.

There is a tradeoff between our two algorithms: although `DTrack` achieves better results (lower overall costs), it has a higher computational time complexity, and requires discovering the hold costs of a large number of servers every time unit. In contrast, `CTrack` has a constant per-unit time complexity, and does not need to probe other servers for their costs except when it decides to transition.

In Section 5.1, we propose two motion-aware heuristic algorithms, named `TargetAware` and `DirectionAware`. `TargetAware` assumes knowledge of the mobile node's current target and speed, whereas `DirectionAware` only requires the knowledge of the node's current direction, which is used to estimate the target, and speed. These hints can be received either from a higher-level application, or from a positioning system like GPS. Although their lookahead window is quite small (the node's next target), both motion-aware algorithms yield significant cost improvements. Their costs are typically within 10% of the optimal, and exhibit perfect scalability.

1.1 Related Work

The nomadic server problem is reminiscent of a number of previously studied online optimization problems.

In the classical *k-server problem* [5], k identical mobile *servers* reside in a metric space M, and have to serve a sequence of requests. When a request is received at point p , one of the servers must move to p in order to service the request. This problem fundamentally differs from ours, as the model model does not include setup and hold costs, and these cannot be reduced to a metric space with mobile servers. Consequently, the algorithmic techniques used to tackle these problems are very different.

The problem of dynamic session management was studied in the context of routing virtual circuits in mobile communication systems [3], with a similar model of setup and hold costs. However, these costs were defined per link, and the algorithm had to decide whether to retain or to release a redundant link. In contrast, in our model, the costs are defined per server, which leads to very different results.

Optimal center location for a group of users is an instance of the well-studied *facility location problem* [13], which given a set of facility locations and a set of customers in a metric space, chooses which customers should be served from which facilities so as to minimize the total service cost. Facility location was studied as an online problem [12], and was used for various applications, including optimizing the delivery of Web content in CDN's [8, 14], maintenance of mobile centers in ad-hoc networks [4] and adaptive server selection in online games [11]. The problem differs from ours in that multiple facilities are used per group, and the online algorithm is allowed to *add* facilities over time, instead of migrating sessions among existing ones.

Balazinska et al. [2] present a nomadic server architecture in the context of a loosely-coupled distributed system for stream query processing. The optimization goal was to achieve load balancing between nodes subject to bilateral agreements. This paper did not consider the existence of migration costs.

2 System Model

Consider an application session that can be hosted by any one of k servers $\mathcal{S} = \{s_0, ..., s_{k-1}\}$. The session is assigned to some server at the beginning of the session but can be re-assigned to a different server at each discrete time slot.

There are two types of non-negative cost charged for the session: a *setup cost* that is paid when the session is assigned to a new server, including the initial one, and a *hold cost*, paid for each time slot the session is assigned to some server. From a session's perspective, different servers offer different costs at a given time slot, and may also change them at the beginning of each slot. We denote the setup cost offered by server s at time t by $\texttt{setup}(s,t)$ and the hold cost by $\texttt{hold}(s,t)$.

The *assignment schedule* $\sigma(t)$ in a time interval $\mathcal{I}$ is a function, $\sigma : \mathcal{I} \to \mathcal{S}$, which assigns the session to server $s \in \mathcal{S}$ at each discrete time $t \in \mathcal{I}$. For convenience, we define $\sigma(t) = \perp$ for $t \notin \mathcal{I}$. We define the set of *transitions* on an interval $\mathcal{I}$ as

$$\mathcal{T}(\sigma, \mathcal{I}) = \{t \mid t \in \mathcal{I} \ \wedge \ \sigma(t) \neq \sigma(t-1)\}.$$

In particular, the initial assignment is also considered a transition.

The assignment schedule σ on an interval $[t_1, t_2)$ incurs a total hold cost

$$\texttt{hold}(\sigma, [t_1, t_2)) \triangleq \sum_{t=t_1}^{t_2-1} \texttt{hold}(\sigma(t), t),$$

a total setup cost

$$\texttt{setup}(\sigma, [t_1, t_2)) \triangleq \sum_{t \in \mathcal{T}(\sigma, [t_1, t_2))} \texttt{setup}(\sigma(t), t),$$

and a total overall cost

$$\texttt{cost}(\sigma, [t_1, t_2)) \triangleq \texttt{setup}(\sigma, [t_1, t_2)) + \texttt{hold}(\sigma, [t_1, t_2)).$$

The *optimal nomadic server assignment* problem for interval $[0, T)$ is to compute an assignment schedule σ^* that minimizes $\texttt{cost}(\sigma^*, [0, T))$.

The presence of positive setup costs is what makes the problem nontrivial. Otherwise, the session would always associate with the server that offers the minimum hold cost. Hence, we always consider the positive setup costs.

3 An Optimal Offline Algorithm

In this section, we describe an optimal *offline* algorithm for the assignment problem, i.e., assuming that the setup and hold cost functions are known in advance. The algorithm is linear-time in the interval length T and the number of servers k.

We first identify the structure of the optimal solution σ^*. Let $\sigma^*_{s,t} : [t, T) \to \mathcal{S}$ be a lowest cost schedule among those in which s is the initial assignment, that is, $\sigma^*_{s,t}(t) = s$. We observe that if $\sigma^*_{s,t}(t+1) = s'$, then

$$\texttt{cost}(\sigma^*_{s,t}, [t+1, T)) = \texttt{cost}(\sigma^*_{s',t+1}, [t+1, T)).$$

In other words, the cost of an optimal schedule for $[t+1, T)$ that assigns s' at $t+1$ is identical to the cost of the $[t+1, T)$-suffix of the optimal schedule for $[t, T)$ with the same assignment. Otherwise, the global optimality is violated. If $s' = s$, then $\texttt{setup}(s', t+1)$ does not contribute to $\texttt{cost}(\sigma^*_{s,t}, [t, T))$. We define the *tail contribution* function for $t < T$ as follows:

$$\texttt{tail}(s, s', [t, T)) \triangleq \begin{cases} \texttt{cost}(\sigma^*_{s,t}, [t, T)) - \texttt{setup}(s', t) & \text{if } s = s' \\ \texttt{cost}(\sigma^*_{s,t}, [t, T)) & \text{otherwise} \end{cases}$$

Then, $\texttt{cost}(\sigma^*_{s,t}, [t, T))$ for $t < T$ can be expressed as

$$\texttt{cost}(\sigma^*_{s,t}, [t, T)) = \texttt{setup}(s, t) + \texttt{hold}(s, t) + \min_{s' \in \mathcal{S}} \texttt{tail}(s, s', [t+1, T))$$

We define $\texttt{tail}(s, s', [T, T)) = \texttt{cost}(\sigma^*_{s,t}[T, T)) = 0$. For $t < T$ we get:

$$\begin{aligned} &\texttt{cost}(\sigma^*_{s,t}, [t, T)) = \\ &\quad \texttt{setup}(s, t) + \texttt{hold}(s, t) + \\ &\quad \min(\min_{s' \in \mathcal{S}} \texttt{cost}(\sigma^*_{s',t+1}, [t+1, T)), \texttt{cost}(\sigma^*_{s,t+1}, [t+1, T)) - \texttt{setup}(s, t+1)). \end{aligned}$$

An optimal solution can be computed through dynamic programming [6] using the above recurrence. The algorithm employs a two-dimensional table $\texttt{Table}[1..k, 0..T]$ where an entry $\texttt{Table}[s, t]$ holds the value of $\texttt{cost}(\sigma^*_{s,t}, [t, T))$ and the identity of $s' = \sigma^*_{s,t}(t+1)$. The table is computed column by column from $T-1$ down to 0. Column T is initialized by zeroes. During the processing of column t, the value of

$$\min_{s' \in \mathcal{S}} \texttt{cost}(\sigma^*_{s',t}, [t, T)) = \min_{1 \le s \le k} \texttt{Table}[s, t]$$

is computed once to be used in computing all entries of column $t-1$. After the whole table is filled, the overall optimal cost is computed as

$$\texttt{cost}(\sigma^*, [0, T)) = \min_{0 \le s \le k-1} \texttt{Table}[s, 0],$$

and an optimal schedule is built by tracing the algorithm's choices through the columns $0 \ldots T-1$.

The computation of a single table entry requires a constant number of operations thanks to the pre-computation of the previous column's minimum cost, and therefore, the algorithm's time complexity is $O(kT)$. The space complexity is also $O(kT)$ – as the table's size.

4 Online Server Assignment

In a realistic scenario, the costs are not known in advance. This is especially true for the hold cost, which can reflect dynamic network conditions like user mobility, group membership, etc. In this section, we study server assignment as an *online* optimization problem [5]. The cost for a time slot becomes known at the beginning of that slot, and the algorithm must produce a new scheduling decision. We restrict ourselves to the case where the setup costs are identical and constant, that is, $\texttt{setup}(s,t) = C$ for all s and t, whereas the hold costs are dynamic. We denote the schedule produced by the optimal algorithm OPT as σ^*, and the schedule produced by an online algorithm ALG as σ.

The *competitive ratio* is the common performance measure for online algorithms. In our problem, an online algorithm ALG is called $r(\texttt{ALG})$-*competitive* if there is a constant δ such that for *all* finite intervals $\mathcal{I}$ and for *all* setup and hold costs

$$\texttt{cost}(\sigma^*,\mathcal{I}) \leq r(\texttt{ALG}) \cdot \texttt{cost}(\sigma,\mathcal{I}) + \delta.$$

The rest of this section is structured as follows. In Section 4.1, we show that no deterministic online algorithm can achieve a competitive ratio better than k. In Section 4.2, we present a generic online algorithm called DTrack (deficit tracker). A version of this algorithm termed DTrack − RR, that is, DTrack with round-robin selection of server assignments, achieves a competitive ratio of $2k$ with a certain parameter choice. DTrack needs to track the cost of up to k servers every time slot, and may thus have a large control message overhead in a distributed implementation. In Section 4.3, we present a simple and efficient algorithm called CTrack (cost tracker), which yields a competitive ratio of $(2+a)k$ for a certain parameter choice, assuming that a server's per-slot hold cost never exceeds aC. The competitive version of CTrack, called CTrack−RR, probes the cost of only one server every slot. In Section 4.4, we present opportunistic versions of these algorithms, called CTrack−F, DTrack−F, and DTrack−B, which greatly improve the cost in the *average* case, and achieve good scalability.

4.1 A Lower Bound of k on the Competitive Ratio

Theorem 1 *The lower competitive ratio of any deterministic server assignment algorithm is at least k.*

Proof : Consider k symmetric servers that offer the same setup cost $C > 0$ and a zero hold cost each at $t = 0$, that is, $\texttt{hold}(s_i, 0) = 0$. Consider the following simple adversary strategy against any deterministic algorithm ALG. When ALG connects to s_i at time t, set $\texttt{hold}(s_i, t+1) = 1$. When ALG disconnects from the server at time t', set $\texttt{hold}(s_i, t'+1) = 0$. Regardless of what the online algorithm is, it will have to transition to a different server at some point if it wishes to remain competitive. This process continues until $k-1$ moves happen. At this point, the adversary stops the run.

If ALG has visited every server exactly once, let s^* be its last assignment. Otherwise, there exists a server s^* that has never been picked by ALG. The best offline algorithm, OPT, assigns the session to server s^* at time 0 and never changes the assignment.

OPT pays only C for the initial setup, whereas ALG pays kC for setup and zero or more for hold. Therefore, $r(\texttt{ALG}) \geq \frac{kC}{C} = k$, and the algorithm's competitive ratio has a lower bound of k. □

4.2 DTrack - a $2k$-Competitive Online Algorithm

We present a simple online algorithm called DTrack (*deficit tracker*). It is parameterized by factor $\alpha \geq 0$, which controls *when* transitions happen, and a subroutine $\texttt{nextchoice}()$, which controls *which* server is chosen upon transition. In this section, we focus on a $2k$-competitive version of DTrack, called DTrack−RR, obtained by a round-robin nextchoice() policy. Its pseudocode appears in Figure 1.

We begin with some definitions. The *deficit* between the servers s and s' during the interval $[\tau, t)$ is the greatest total difference between the total hold costs in a suffix $[t', t)$:

$$\texttt{def}(s, s', [\tau, t)) \triangleq \max_{\tau \leq t' \leq t-1} (\texttt{hold}(s, [t', t)) - \texttt{hold}(s', [t', t))).$$

Let us denote the current assignment by s_c. A server s for which $\texttt{def}(s_c, s, [\tau, t+1)) > 0$ is called a *leader* at time t.

DTrack maintains an invariant that the deficit between s_c and any other server s never exceeds αC. Initially, DTrack makes an assignment to the server with the minimal hold cost. It then keeps tracking the deficit versus the other servers. A server becomes a leader when it offers a smaller hold cost than s_c, and stops being one when the accumulated deficit value becomes negative. Since the hold costs are published at the beginning of each time slot, DTrack makes its decision using a single-slot lookahead. When some server is about to accumulate significantly less hold cost than the current choice (a deficit of above αC), the algorithm changes its assignment. Due to the lookahead mechanism, the update() procedure that updates the deficit values is invoked twice at transition times. First, for the current choice in order to decide whether to transition, and then for the new choice, which does not necessarily offer the best hold cost, hence the new deficit must be computed.

In the instance of DTrack we consider now, termed DTrack−RR, nextchoice() selects the next assignment in a round-robin way, among servers whose a-priori deficit versus any other server (that is, the hold cost gap) does not exceed αC.

The intuition behind DTrack is that the current server must be provably bad (costing αC more than the best) in order to change the choice, and the next server must also *not* be provably bad (not costing αC more than any other server). When instantiated with $\alpha = 0$ (this algorithm is termed Greedy), DTrack immediately changes the assignment when some other server offers a better hold cost. At the other extreme, when $\alpha = \infty$, it never changes its initial assignment. It is clear that the algorithm is not competitive in either of these extreme cases.

In Appendix A.1, we provide a detailed competitive analysis of DTrack−RR, and get the following result:

Theorem 2 *The competitive ratio of* DTrack−RR *is bounded as follows:*

$$\begin{array}{ll} r(\texttt{DTrack}-\texttt{RR}) < \texttt{k}(1 + \frac{1}{\alpha}) & \alpha \leq 1 \\ r(\texttt{DTrack}-\texttt{RR}) < 1 + (\texttt{k}-1)\alpha + \texttt{k} & \alpha \geq 1 \end{array}$$

Corollary 1 *For $\alpha = 1$,* DTrack−RR *achieves a competitive ratio of $2k$.*

The crux of the algorithm's competitiveness lies in the round-robin selection policy, and can be informally explained as follows. If we consider a schedule σ by DTrack−RR that either *overtakes* (that is, either leaves or skips) every server while the optimal schedule σ^* does not change its assignment s^*, then σ overtakes s^* exactly once. This overtake implies that the total hold cost incurred by σ^* during the interval exceeds αC. The total hold cost incurred by σ exceeds the one incurred by σ^* by at most $(k-1)\alpha C$. The subtle point in this proof is the deficit bookkeeping, because upon transition the hold cost lookahead affects the assignment but does not contribute to

Initialization:
  $c \leftarrow i$ s.t. $\texttt{hold}(s_i) = \min_{s \in \mathcal{S}} \texttt{hold}(s)$
  $\texttt{reset}()$

Every time slot do
  $\texttt{update}()$
  if $(\texttt{def}(s_c, s) > \alpha C)$ for some $s \in \texttt{Leaders}$ **then**
    $\texttt{nextchoice}()$
    $\texttt{reset}()$

procedure $\texttt{reset}()$
  $\texttt{Leaders} \leftarrow \emptyset$
  for all $s \neq s_c$ **do**
    $\texttt{def}(s_c, s) \leftarrow 0$
  $\texttt{update}()$

procedure $\texttt{update}()$
  for all s s.t. $s \notin \texttt{Leaders} \wedge \texttt{hold}(s_c) > \texttt{hold}(s)$ **do**
    $\texttt{def}(s_c, s) \leftarrow 0$
    $\texttt{Leaders} \leftarrow \texttt{Leaders} \cup \{s\}$
  for all $s \in \texttt{Leaders}$ **do**
    $\texttt{def}(s_c, s) \leftarrow \texttt{def}(s_c, s) + (\texttt{hold}(s_c) - \texttt{hold}(s))$
    if $(\texttt{def}(s_c, s) < 0)$ **then**
      $\texttt{Leaders} \leftarrow \texttt{Leaders} \setminus \{s\}$

procedure $\texttt{nextchoice}()$ */*RR version*/*
  repeat
    $c \leftarrow (c+1)$ **mod** k
  until $\texttt{hold}(s_c) - \min_{s \in \mathcal{S}} \texttt{hold}(s) \leq \alpha C$

Figure 1: $\texttt{DTrack}-\texttt{RR}$ **- an Online Algorithm for Server Assignment.**

the total hold cost. The total setup cost incurred by σ during this period is at most kC, whereas σ^* pays C upon the assignment to s^*. A careful analysis of the worst-case ratio between the total costs concludes the proof.

At each slot, `DTrack` checks the hold cost of every server, which results in linear time complexity per slot. Since the number of servers can be large, sublinear complexity is desirable to achieve efficiency of communication in a distributed implementation.

4.3 CTrack - an Efficient Online Algorithm

We now present a simple online algorithm `CTrack` (*cost tracker*), which achieves constant computation time complexity at the expense of a weaker competitive guarantee, under the assumption of an upper bound on the ratio between the hold and the setup costs. `CTrack` is also parameterized by a factor α and a subroutine `nextchoice()`. Initially, it assigns the server with the minimal hold cost. The assignment changes when the total hold cost since the last transition exceeds αC (e.g., for $\alpha = 0$, it transitions every time slot). The rationale behind this policy is controlling the fraction of the setup cost in the total cost. It only requires receiving the hold cost of the *current* assignment every time slot, which leads to constant per-slot time complexity.

In Appendix A.2, we provide a detailed competitive analysis of `CTrack−RR`, the round-robin version of `CTrack`, and get the following result:

Theorem 3 *If* $\texttt{hold}(s,t) \le aC$ *for all* s *and* t*, then* $r(\texttt{CTrack−RR}) < (2+a)k$ *for* $\alpha = 1$*.*

4.4 Opportunistic Heuristics

While the competitive ratio is an accepted metric for measuring the worst-case performance of an online algorithm, the average-case performance is more important in practice. An algorithm that behaves $2k$ times worse than the optimal solution in the average case is impractical in systems accommodating thousands of servers.

In this section, we introduce opportunistic versions of `CTrack` and `DTrack`, in which `nextchoice()` selects an assignment that is locally optimal for some metric, instead of the round-robin traversal. This approach exploits the well-known locality principle to achieve good performance in typical scenarios. Note that although locality is common in practice, it is not a property that holds in all possible runs, and hence, the cost of using opportunistic selection policies is that they yield worse competitive ratios than the round-robin ones.

In the *forward* heuristics `DTrack−F` and `CTrack−F`, `nextchoice()` picks the server with the current minimal hold cost. The *backward* heuristic `DTrack−B` augments `DTrack−RR`'s selection policy with the following rule: the deficit between the next choice and the previous assignment is greater than βC for some $-\infty \le \beta \le \alpha$. Using any $\beta > 0$ allows the algorithm to choose the next server from those that presented good behavior since the last transition. For $\beta = -\infty$, the resulting algorithm is `DTrack−RR`. For $\beta = 0$, `DTrack−B` chooses the next server from the leader set. For $\beta = \alpha$, it selects a leader that triggered the transition. Theorem 2 can be generalized to describe `DTrack−B`'s worst-case behavior (the proof appears in Section A.3):

Theorem 4 *The competitive ratio of* `DTrack−B` *is bounded as follows:*

$$\begin{array}{ll} r(\texttt{DTrack}) < \texttt{k}(\texttt{1} + \frac{1}{\alpha}) & \alpha \le 1 \text{ and } \beta \le 0 \\ r(\texttt{DTrack}) < \texttt{1} + (\texttt{k}-\texttt{1})\alpha + \texttt{k} & \alpha \ge 1 \text{ and } \beta \le \alpha - 1 \\ r(\texttt{DTrack}) < \texttt{1} + \frac{(\texttt{k}-1)\alpha+\texttt{k}}{\alpha-\beta} & \max(0, \alpha-1) \le \beta \le \alpha \end{array}$$

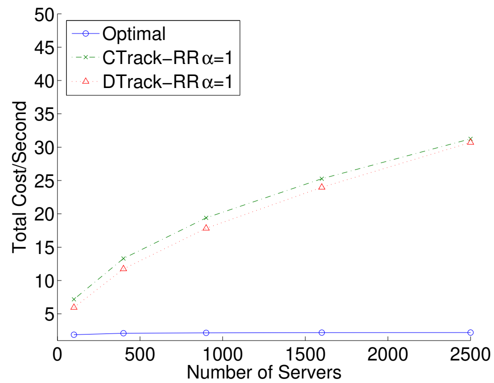


Figure 2: **CTrack−RR and DTrack−RR with $\alpha = 1$ do not scale well with the network size.**

Corollary 2 *For $\alpha = 1$ and $\beta \leq 0$,* `DTrack−B` *achieves a competitive ratio of $2k$.*

The worst-case competitive ratio achieved by `DTrack−B` with $\alpha = \beta$ is not limited by the problem size k (see Appendix A.4 for the proof):

Theorem 5 *The competitive ratio of* `DTrack−B` *with $\alpha = \beta$ is $\Omega(C)$.*

5 Case Study: Mobile Users in a WMN

In this section, we study nomadic server assignment in an urban WMN environment. The results of the optimal algorithm `OPT` are used as a comparison baseline. For each algorithm `ALG`, we measure its cost as well as *performance ratio*, which is the average ratio between the total costs incurred by `ALG` and `OPT` during multiple runs. We average over 20 simulations, each 10,000 slots long. This metric is analogous to the competitive ratio, the theoretical worst-case metric.

The simulated network spans a square grid with uniformly distributed wireless routers. The number of routers that populate a 1000m × 1000m grid is 100, that is, a single router spans an average area of 100m×100m. A mobile node moves using the random waypoint mobility model [15]. The node uniformly chooses the destination and moves toward it at a constant urban driving speed of 10 m/sec (36 km/hour). The time slot is one second. The hold cost between mobile node n and router r is defined as $\frac{d(n,r)}{100}$, i.e., a normalized Euclidean (L_2) distance. Under these parameters, the average hold cost offered by the closest router is roughly 0.5. The setup cost is 50.

Our main interest is in the scalability of the online solutions, i.e., how the total cost per second and the performance ratio are affected as the problem size grows. For this purpose, we gradually increase the grid size from 1000m × 1000m to 5000m × 5000m, and correspondingly increase the number of routers from 100 to 2500, keeping the router density fixed. We study the performance of different versions of `CTrack` and `DTrack` with different selections of α, β, and `nextchoice`().

Our first goal is to study the performance of `CTrack−RR` and `DTrack−RR` with $\alpha = 1$, which have the best proven worst-case ratios. Figure 2 shows that both algorithms scale poorly with the network size (their costs grow approximately as $\sqrt{k}$, whereas `OPT`'s cost remains nearly constant). This is intuitive, since the round-robin selection policy tends to assign a session to a random server, and the average distance grows as $O(\sqrt{k})$.

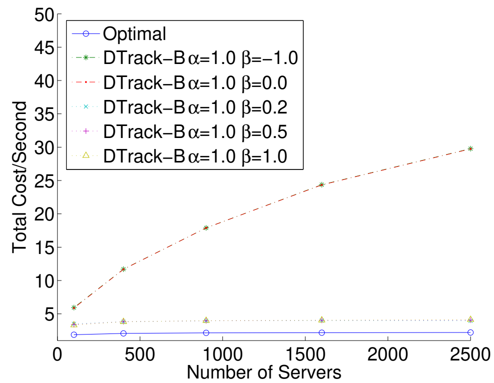


Figure 3: **Choosing a β value for DTrack−B with $\alpha = 1.0$. The values between 0.2 and 1 exhibit very close behavior and scale well with the network size.**

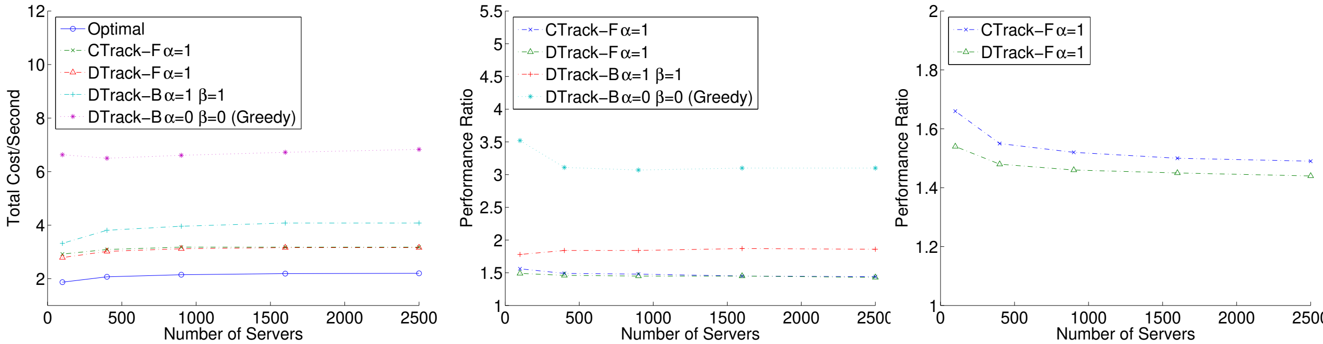
DTrack−B requires selecting the β parameter for a given α. Contrary to the worst-case analysis, our results show that the algorithm's performance improves as β becomes closer to α. Figure 3 depicts the results for $\alpha = 1$. The curves for all β values from 0.2 to 1 are barely distinguishable. Hence, a good worst case ratio can be guaranteed by selecting small β values without compromising the average performance by much (for example, for $\alpha = 1$ and $\beta = 0.2$, the competitive ratio is bounded by $2.5k - 0.25$).

Figures 4(a) and 4(b) depict the results of simulating the opportunistic algorithms Greedy, CTrack−F, DTrack−F, and DTrack−B with $\alpha = 1$ and $\beta = 1$. The performance curves of CTrack−F and DTrack−F are almost indistinguishable. The algorithms' performance ratios remain constant as the problem scales – around 50% above the optimum. The total cost per second also remains constant, since OPT itself is very scalable. Greedy, which takes the opportunistic heuristic to the extreme, exhibits a weaker performance ratio (more than three times the optimum) although it scales well. In this setting, Greedy's reasonable behavior can be explained by the moderate speed (hence, the hold cost changes are slow), and by the moderate setup cost (hence, the penalty for making a wrong decision is limited). The fact that DTrack−F consistently produces better results than DTrack−B can be explained by the motion's nature. Since the motion is random, the deficit values exhibit poor locality. The result could have been different had the motion happened around a small number of stationary points (home, office, cab station etc).

Figure 4(c) depicts the results of the same experiment with an average simulated speed 25 m/sec (90 km/hour). In this setting, DTrack−F starts producing a consistently lower total cost (by 5-6%) than CTrack−F. This happens because at higher speeds, the hold cost changes faster, and the total cost becomes a worse transition indicator than the deficit. This phenomenon cannot be further magnified at reasonable driving speeds, but can be clearly demonstrated in a different application (Section 6). As expected, Greedy performs worse at higher speeds (above five times the optimum).

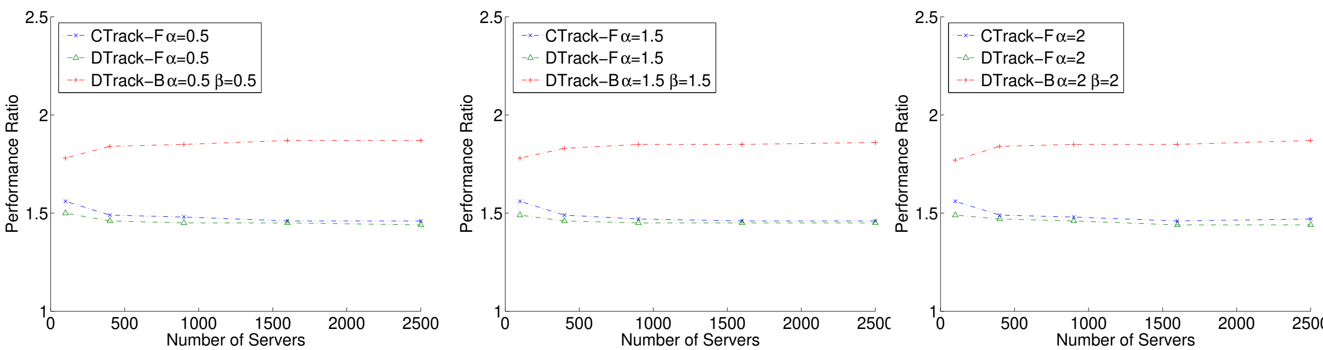
Further simulations(Figure 5) show that α values between 0.5 and 2.0 exhibit nearly the same average-case performance.

DTrack's computation overhead can be significantly improved in a WMN environment since the hold cost monotonically increases with distance. Therefore, maintaining the deficit values requires accessing the hold costs of the servers that are closer to the user than the current assignment, as well as the servers that already have a positive deficit. This can be achieved by using data



(a) Average total cost/second, speed=10 m/s (b) Performance ratio, speed=10 m/s (c) Performance ratio, speed=25 m/s

Figure 4: **Scalability of `CTrack−F`, `DTrack−F`, and `DTrack−B` in a WMN with mobile users, speed=10 m/s, $\alpha = 1.0$ and $\beta = 1.0$.**



(a) Performance ratio, $\alpha = 0.5$ (b) Performance ratio, $\alpha = 1.5$ (c) Performance ratio, $\alpha = 2.0$

Figure 5: **Scalability of `CTrack−F`, `DTrack−F`, and `DTrack−B` in a WMN with mobile users, speed=10 m/s, with different α values.**

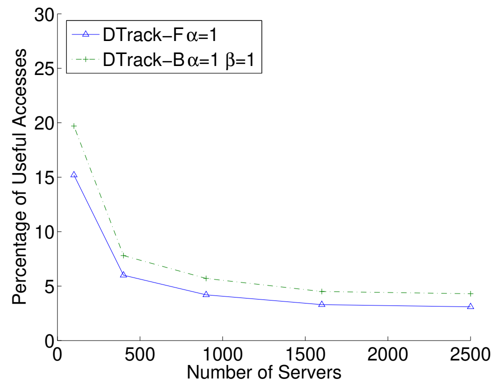


Figure 6: **Percentage of useful hold cost accesses per second for `DTrack−F` and `DTrack−B` with $\alpha = 1$ and $\beta = 1$.**

structures that support efficient nearest neighbor queries in a multidimensional space like KD-trees or R-trees [13]. Figure 6 depicts the percentage of hold costs that need to be accessed by `DTrack−F` and `DTrack−B` with $\alpha = \beta = 1$. We can see that the fraction of hold costs that must be accessed to maintain the positive deficit values is very low.

5.1 Motion-Aware Heuristics

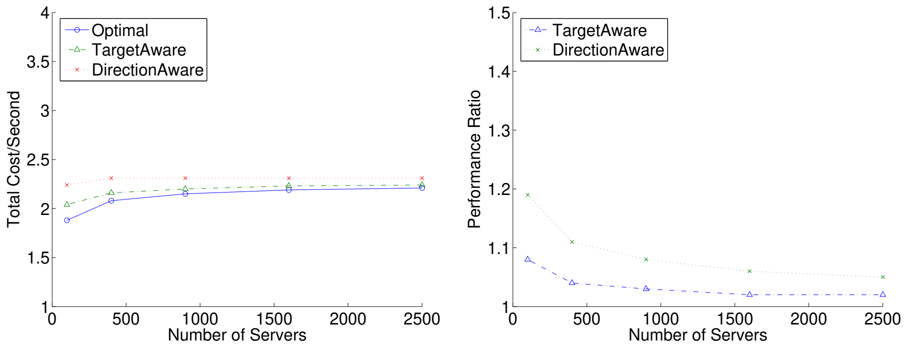
In order to achieve a better practical performance, we employ two simple online heuristics tailored specifically to the mobile user environment. These heuristics exploit the near-term motion pattern, and therefore can project the hold costs better than `DTrack`, which has only a single-slot lookahead.

The first heuristic is called `TargetAware`. It requires information regarding the mobile node's current target and speed. This target information can be provided from a higher-level system, e.g., a car navigation system, where the user can indicate the current status (e.g., "driving home"). `TargetAware` is informed every time the mobile node changes its target, and applies `OPT` as a subroutine in order to compute the assignment schedule until the next target is reached. Every time the target changes, `TargetAware` selects the best of two choices: running `OPT` with the fixed first assignment that is identical to the current one (i.e., no setup cost is incurred for it), or letting `OPT` pick an arbitrary first assignment.

If the target information is not available, a mobile node equipped with a positioning system (e.g., GPS) can use the direction information provided by it. In this context, we propose the second heuristic that is called `DirectionAware`. It receives information about the grid size as well as the mobile node's current direction and speed, which are received upon the node's direction changes. The algorithm projects the next target as the clipping point of its current trajectory and the grid's boundary, and applies `TargetAware` as a subroutine.

Figure 7 depicts the scalability of both motion-aware heuristics, in the same environment as the previous simulation. Both `TargetAware` and `DirectionAware` are clearly superior to `CTrack−F` and `DTrack−F`. Their performance ratios are less than 10% and 20% above the optimum, respectively. As expected, `TargetAware` performs slightly better than `DirectionAware` because it uses an accurate motion forecast. The motion-aware heuristics scale even better than `OPT` because their lookahead window grows as the grid scales up.

Note that both heuristics perform very well despite their small lookahead window. In the context



(a) Average total cost/second, speed=10 m/s (b) Performance ratio, speed=10 m/s

Figure 7: **Scalability of the motion-aware algorithms in a WMN with mobile users.**

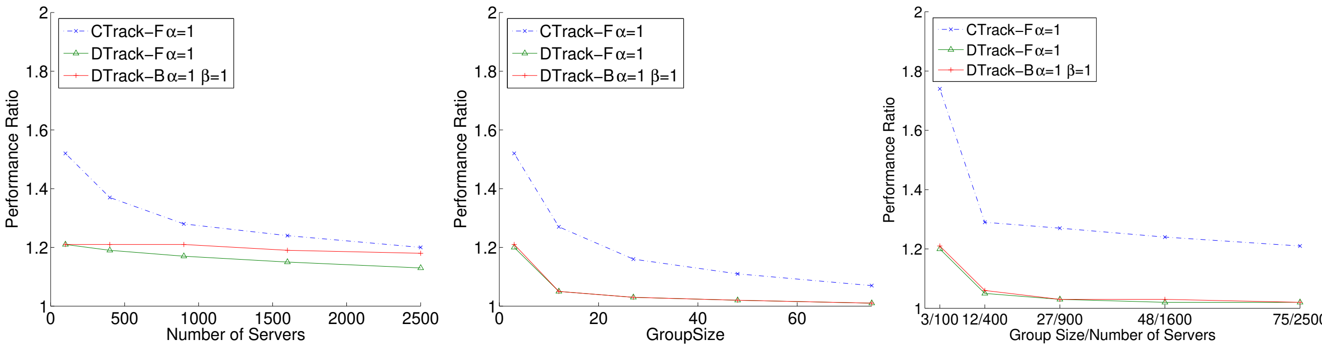
of the offline assignment problem, this means that a practically good solution can be achieved with constant space complexity, without the need to capture the entire data stream before running the dynamic programming algorithm.

6 Case Study: Wide-area Chatroom Service

The second environment studied is an Internet-scale groupware application service [10, 11], e.g., chat. The service overlay network consists of 100 servers uniformly selected among the nodes of a random network. Groups of users run a chatroom application, where each group is assigned to a single server. The users are stationary, and their locations are uniformly distributed in the network. The user arrival to a group is described by a Poisson process with a mean of λ, and the membership lifetime is distributed exponentially with a mean of T (that is, the average number of users in a group is λT). The hold cost between group G and server s is proportional to the maximal network distance between the server and some node in the group, which reflects the application's buffer space requirements affected by the maximal delay. In this context, the server is seen as the group's *center*, and the maximal distance is the group's *radius*. We study the same instances of `CTrack−F`, `DTrack−F`, and `DTrack−B` as in Section 5 (that is, $\alpha = \beta = 1$). We explore the algorithms' scalability with both the number of servers and the average group size.

In the first experiment, we increase the number of servers (in parallel with the network's size) from 100 to 2500, without increasing the number of users. We set $\lambda = 0.1$ users/second and $T = 30$ seconds, yielding three users in the chatroom on average. Figure 8(a) depicts the simulation results. Both versions of `DTrack` are within 15-20% above the optimal cost. `DTrack−F` consistently outperforms `CTrack−F` because individual join or leave events in a small group trigger fast changes in the hold costs. This is the same phenomenon that happens in WMNs at high speeds (Figure 4(c)), but it is more significant since the hold cost changes are faster.

In the second experiment, depicted in Figure 8(b), we scale the average group size up from three to 75 (a large-scale conference) by increasing both λ and T. The network size is not changed. Both versions of `DTrack` exhibit a performance ratio of under 5% above the optimum for groups with more than ten members, and converge to the optimal cost as the group scales. This happens because in dense groups, individual join and leave events do not considerably affect the group radius. Therefore, the algorithms perform fewer transitions.



(a) Scalability with number of servers (b) Scalability with group size (c) Scalability with number of servers/group size

Figure 8: **Scalability of `CTrack−F`, `DTrack−F` and `DTrack−B` in a wide-area chatroom application service, $\alpha = 1.0$ and $\beta = 1.0$.**

Finally, we study the algorithms' scalability to large groups in large networks. For this purpose, we gradually increase both the number of servers and the group size by the same factor. The results depicted in Figure 8(c) show that when the number of servers grows from 400 to 2500 and the number of users grows from 12 to 75, the performance ratios of both versions of `DTrack` remain constant at less than 5% above the optimum, whereas the performance ratio of `CTrack−F` also remains constant but exceeds the optimum by 30%.

7 Conclusion

In this paper, we have studied a novel problem of service point assignment to mobile users or user groups in a distributed infrastructure with multiple service points. This problem will naturally arise in several emerging practical environments. We have provided a rigorous theoretical study, which includes competitive online algorithms and a lower bound on the competitive ratio of deterministic algorithms. Following this, we studied the performance of the proposed algorithms when applied in an urban WMN and in a wide-area chatroom service. We gave practical algorithms that exhibit good performance relative to the optimal solution and scale well with the network size.

Acknowledgement

The authors would like to thank Seffi Naor for stimulating discussions.

A Competitive Analysis

A.1 A Competitive Analysis of DTrack-RR

In this section, we give a competitive analysis of the worst-case performance of `DTrack−RR`, and derive the parameter value of α for which the best competitive ratio is obtained. We first prove that the `nextchoice`() subroutine always succeeds finding a new assignment.

Lemma 1 $\texttt{def}(\sigma(t), s) \leq \alpha C$ *for all* s *and* t.

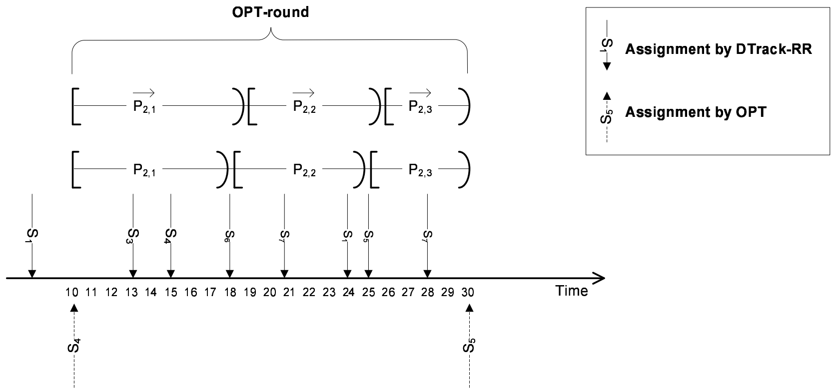


Figure 9: **Definition of phases for `DTrack−RR`.**

Proof : By induction on t. For $t = 0$, the claim holds because the server with the minimal hold cost is selected. For $t > 0$, if no transition happens at time t, then the algorithm maintains the invariant. Otherwise, assume a transition happened at time t, and consider the next transition from $s_c = \sigma(t)$ at time $t' \geq t+1$. By definition, $s_c = \sigma(t'-1)$. Hence, $\texttt{def}(s_c, s, [t, t')) \leq \alpha C$ for all s by induction hypothesis. However, for some s, $\texttt{def}(s_c, s, [t, t+1)) > \alpha C$. Therefore, there exists some server s such that $\texttt{hold}(s, t) > \texttt{hold}(s_c, t)$. Hence, $\texttt{hold}(s_c, t)$ is not the minimal hold cost at time t, that is, some identity $s \neq s_c$ can be found such that $\texttt{def}(s, s', [t', t+1)) \leq \alpha C$, for all s' (for example, the server with the minimal hold cost satisfies this requirement). □

Corollary 3 *If* `nextchoice`() *is invoked at time t, it returns an identifier that is different from* $\sigma(t-1)$.

We term an interval $[\tau, \tau')$ between two consecutive transitions of algorithm `ALG` or between `ALG`'s last transition and the end of the run as `ALG`-*round*. Where the `ALG` is clear from the context, we simply say round.

It is convenient to describe the assignment choices made by `DTrack−RR` with time as a movement in a circular server identifier space, with a clockwise direction from s to $(s+1) \bmod k$. We say that σ *overtakes* s at time t if s is encountered while moving clockwise from $\sigma(t-1)$ to $\sigma(t)$, and $s \neq \sigma(t)$. In other words, either $\sigma(t-1)$ is s, or s is skipped at t.

We now consider a `DTrack−RR`-round and an `OPT`-round. We analyze the competitive ratio of `DTrack−RR` for different values of α by comparing the cost it incurs with the cost incurred by `OPT` during a single `OPT`-round $[\tau_i, \tau_{i+1}]$ and then generalizing for the whole run. We denote `OPT`'s assignment during this `OPT`-round by s^*.

We define two partitions of the interval $[\tau_i, \tau_{i+1})$ into sub-intervals. The first one partitions the interval to *phases* $\{\mathcal{P}_{i,j} = [t_{i,j}, t_{i,j+1})\}$, defined as follows. The first phase starts at τ_i. A phase completes at the earlier between the time when σ overtakes s^* and τ_{i+1}. The second partition is to *shifted phases* $\{\overrightarrow{\mathcal{P}_{i,j}}\}$, defined as follows. The first shifted phase starts at τ_i. A shifted phase completes at the earlier between one slot after the completion of the corresponding phase and τ_{i+1}. The number of phases and shifted phases is equal by definition.

Figure 9 depicts the above definitions for an `OPT`-round $[10, 30)$, in which $S^* = S_4$. The first phase ends at time 18 when the algorithm chooses S_6 and overtakes S_4, which was its previous assignment. The second phase ends at time 25 when the algorithm chooses S_5 and overtakes S_4 for the second time, without choosing S_4 in this phase.

Lemma 2 *Consider an* `OPT`*-round* $[\tau_i, \tau_{i+1})$ *with* p *phases produced by* `DTrack−RR`. *Then,*

$$\texttt{cost}(\sigma, [\tau_i, \tau_{i+1})) \leq \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) + pC(k + (k-1)\alpha)$$

Proof : Consider a `DTrack−RR`-round $[t, t') \subseteq \mathcal{P}_{i,j}$.

If $\sigma(t) = s^*$, then $\texttt{hold}(\sigma^*, [t, t')) = \texttt{hold}(\sigma, [t, t'))$. Otherwise, the difference in the accumulated hold cost between $s = \sigma(t)$ and every other server s' is bounded by $\texttt{def}(s, s', [t, t'))$, which never exceeds αC. In particular,

$$\texttt{hold}(\sigma, [t, t')) - \texttt{hold}(\sigma^*, [t, t')) \leq \alpha C.$$

There are at most $k-1$ rounds during $\mathcal{P}_{i,j}$ in which the assignment is different from s^*, and hence,

$$\texttt{hold}(\sigma, \mathcal{P}_{i,j}) - \texttt{hold}(\sigma^*, \mathcal{P}_{i,j}) \leq (k-1)\alpha C.$$

`DTrack−RR` performs at most k transitions during $\mathcal{P}_{i,j}$, paying at most kC for setup. Therefore,

$$\texttt{cost}(\sigma, \mathcal{P}_{i,j}) \leq \texttt{hold}(\sigma^*, \mathcal{P}_{i,j}) + (k-1)\alpha C + kC.$$

$\{\mathcal{P}_{i,j}\}$ is a partition of $[\tau_i, \tau_{i+1}]$, and hence,

$$\texttt{cost}(\sigma, [\tau_i, \tau_{i+1})) = \sum_{j=1}^{p} \texttt{cost}(\sigma, \mathcal{P}_{i,j}) \leq$$
$$\sum_{j=1}^{p} \texttt{hold}(\sigma^*, \mathcal{P}_{i,j}) + pC(k + (k-1)\alpha) = \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) + pC(k + (k-1)\alpha).$$

□

Lemma 3 *Consider an* `OPT`*-round* $[\tau_i, \tau_{i+1})$ *with* p *phases produced by* `DTrack−RR`. *Then,*

$$\texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) > (p-1)\alpha C.$$

Proof : If $p = 1$, the claim follows immediately because the hold costs are non-negative.

Otherwise, consider a phase $\mathcal{P}_{i,j}$ such that $j < p$. This phase ends at $t_{i,j+1}$ that is strictly smaller than τ_{i+1}. We first prove a claim that $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \alpha C$. Consider `DTrack−RR`'s assignment s during the last `DTrack−RR`-round $[t, t_{i,j+1})$ in $\mathcal{P}_{i,j}$, that is, $s = \sigma(t)$, and σ overtakes s^* at time $t_{i,j+1}$. By definition, $\overrightarrow{\mathcal{P}_{i,j}}$ ends at time $t_{i,j+1} + 1$. Consider two possible cases:

1. If $s \neq s^*$, then the algorithm considers picking s^* upon the transition from s at $t_{i,j+1}$, and does not select it because there exists a server s' s.t. $\texttt{hold}(s^*, t_{i,j+1}) - \texttt{hold}(s', t_{i,j+1}) > \alpha C$, and hence, $\texttt{hold}(s^*, t_{i,j+1}) > \alpha C$. By definition of a shifted phase, $[t_{i,j+1}, t_{i,j+1} + 1) \subseteq \overrightarrow{\mathcal{P}_{i,j}}$. It follows that $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \alpha C$, and the claim holds.

2. Otherwise, $s = s^*$. Since the algorithm transitions from s^* at time $t_{i,j+1}$, there exists s' such that $\texttt{def}(s^*, s', [t, t_{i,j+1}+1)) > \alpha C$, that is, $\texttt{hold}(\sigma^*, [t, t_{i,j+1}+1)) > \alpha C$. If $\mathcal{P}_{i,j}$ is the first phase in $[\tau_i, \tau_{i+1})$, then $[t, t_{i,j+1}+1) \subseteq \overrightarrow{\mathcal{P}_{i,j}}$ by definition of shifted phase. Otherwise, consider the preceding phase $\mathcal{P}_{i,j-1}$. By definition, σ overtakes s^* at time $t_{i,j}$. In particular, $\sigma(t_{i,j}) \neq s^*$. Since at least one time slot is spent at every assignment, σ transitions to s^* at time $t_{i,j} < t < t_{i,j+1}$, that is, $[t, t_{i,j+1}+1) \subseteq \overrightarrow{\mathcal{P}_{i,j}}$. It follows that $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \alpha C$, and the claim holds.

It follows that $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \alpha C$. $\{\overrightarrow{\mathcal{P}_{i,j}}\}$ is a partition of $[\tau_i, \tau_{i+1})$, and therefore,

$$\texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) \geq \sum_{j=1}^{p-1} \texttt{hold}(\sigma, \overrightarrow{\mathcal{P}_{i,j}}) > (p-1)\alpha C.$$

□

Theorem 2 *The competitive ratio of* DTrack−RR *is bounded as follows:*

$$\begin{array}{ll} r(\texttt{DTrack}-\texttt{RR}) < \texttt{k}(1+\frac{1}{\alpha}) & \alpha \leq 1 \\ r(\texttt{DTrack}-\texttt{RR}) < 1 + (\texttt{k}-1)\alpha + \texttt{k} & \alpha \geq 1 \end{array}$$

Proof : Consider the local ratio between the costs incurred by DTrack−RR and OPT during a single OPT-round $[\tau_i, \tau_{i+1})$, that is, $\frac{\texttt{cost}(\sigma,[\tau_i,\tau_{i+1}))}{\texttt{cost}(\sigma^*,[\tau_i,\tau_{i+1}))}$. OPT pays the setup cost C for a single transition during $[\tau_i, \tau_{i+1})$ (at τ_i), and therefore,

$$\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1})) = C + \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})).$$

Substituting the ratio's numerator from Lemma 2, we receive

$$\frac{\texttt{cost}(\sigma, [\tau_i, \tau_{i+1}))}{\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1}))} \leq \frac{\texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) + pC((k-1)\alpha + k}{C + \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1}))} < 1 + \frac{pC((k-1)\alpha + k)}{C + \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1}))}.$$

Substituting the denominator from Lemma 3,

$$\frac{\texttt{cost}(\sigma, [\tau_i, \tau_{i+1}))}{\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1}))} < 1 + \frac{pC(k + (k-1)\alpha)}{C + (p-1)\alpha C} = 1 + \frac{p((k-1)\alpha + k)}{1 + (p-1)\alpha}.$$

We denote $\varrho(\alpha, p) \triangleq 1 + \frac{p(k+(k-1)\alpha)}{1+(p-1)\alpha}$. In order to compute p that produces the maximum ratio for a given α, we derive $\frac{\partial \varrho}{\partial p}$. We get that $\frac{\partial \varrho}{\partial p} = 0$ for $\alpha = 1$, that is, the function is constant: $\varrho(1, p) = 2k$ for all p. The derivative is strictly positive for $\alpha < 1$ and strictly negative for $\alpha > 1$, therefore, the function is monotonically increasing for $\alpha < 1$ and monotonically decreasing for $\alpha > 1$. For $\alpha < 1$,

$$\sup_{1 \leq p < \infty} \varrho(\alpha, p) = \lim_{p \to \infty} \varrho(\alpha, p) = 1 + \frac{(k-1)\alpha + k}{\alpha} = k(1 + \frac{1}{\alpha}),$$

whereas for $\alpha > 1$,

$$\sup_{1 \leq p < \infty} \varrho(\alpha, p) = \varrho(\alpha, 1) = 1 + (k-1)\alpha + k.$$

Since this upper bound limits the algorithm's competitive ratio for every OPT-round, we conclude the same result for the entire run. □

A.2 A Competitive Analysis of CTrack-RR

Theorem 3 *If* $\texttt{hold}(s,t) \le aC$ *for all* s *and* t, *then* $r(\texttt{CTrack-RR}) < (2+a)k$ *for* $\alpha = 1$.

Proof : Consider an $\texttt{OPT}$-round $[\tau_i, \tau_{i+1})$ with p phases produced by $\texttt{CTrack-RR}$ as defined in Appendix A.1, in which s^* is $\texttt{OPT}$'s choice.

Consider a $\texttt{CTrack-RR}$ round $[t, t')$ in which server s is $\texttt{CTrack-RR}$'s choice. If $t < t'-1$, then

$$\texttt{hold}(\sigma, [t, t')) = \texttt{hold}(\sigma, [t, t'-1)) + \texttt{hold}(s, t') \le \texttt{hold}(\sigma, [t, t'-1)) + aC.$$

$\texttt{hold}(\sigma, [t, t'-1)) \le \alpha C$ since no transition happened at $t'-1$, and hence, $\texttt{hold}(\sigma, [t, t')) \le (\alpha + a)C$. If $t = t'-1$, the same result holds trivially. There are p phases in $[\tau_i, \tau_{i+1})$ and at most k rounds in each phase. Summarizing over all $\texttt{CTrack-RR}$'s rounds, we get

$$\texttt{cost}(\sigma, [\tau_i, \tau_{i+1})) \le pkC + \texttt{hold}(\sigma, [\tau_i, \tau_{i+1})) \le pk(\alpha + a)C + pkC = pk(\alpha + a + 1)C.$$

Consider the last $\texttt{CTrack-RR}$ round $[t, t')$ in phase $\mathcal{P}_{i,j}$ such that $j < p$. By definition, s^* is the algorithm's choice in this round. A transition happens, therefore, $\texttt{hold}(\sigma, [t, t')) > \alpha C$. Hence, $\texttt{hold}(\sigma^*, [t, t')) > \alpha C$. Summarizing over all phases in $[\tau_i, \tau_{i+1})$, we get

$$\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1})) = C + \texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) > (1 + (p-1)\alpha)C.$$

Hence,

$$\frac{\texttt{cost}(\sigma, [\tau_i, \tau_{i+1}))}{\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1}))} < k\frac{p(\alpha + a + 1)}{1 + (p-1)\alpha}.$$

For $\alpha = 1$, this ratio is smaller than $(2+a)k$ for all p. Since this upper bound limits the algorithm's competitive ratio for every $\texttt{OPT}$-round, we conclude the same result for the entire run. □

A.3 A Competitive Analysis of DTrack-B

In this section, we prove the upper bound on the competitive ratio of $\texttt{DTrack-B}$ for arbitrary β values. We first introduce a slight modification to Lemma 3 and its proof.

Lemma 4 *Consider an* $\texttt{OPT}$*-round* $[\tau_i, \tau_{i+1}]$ *with* p *phases produced by* $\texttt{DTrack}$. *Then,*

$$\texttt{hold}(\sigma^*, [\tau_i, \tau_{i+1})) > (p-1)C \min(\alpha, \alpha - \beta).$$

Proof : Like in Lemma 3, we consider a phase $\mathcal{P}_{i,j}$ such that $j < p$, which ends at $t_{i,j+1}$. We first prove a claim that $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \max(\alpha, \alpha - \beta)C$. Consider $\texttt{DTrack-B}$'s assignment s during the last $\texttt{DTrack-B}$-round $[t, t_{i,j+1})$ in $\mathcal{P}_{i,j}$, that is, $s = \sigma(t)$, and σ overtakes s^* at time $t_{i,j+1}$. Consider the case when $s \ne s^*$. This happens for one of two reasons:

1. There exists a server s' such that $\texttt{hold}(s^*, t_{i,j+1}) - \texttt{hold}(s', t_{i,j+1}) > \alpha C$, and therefore, $\texttt{hold}(s^*, t_{i,j+1}) > \alpha C$. $[t_{i,j+1}, t_{i,j+1} + 1) \subseteq \overrightarrow{\mathcal{P}_{i,j}}$, hence, $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > \alpha C$, and the claim follows.

2. $\texttt{def}(s, s^*, [t, t_{i,j+1} + 1)) \le \beta C$. There exists a server s' which triggered the transition, and therefore, $\texttt{def}(s, s', [t, t_{i,j+1} + 1)) > \alpha C$. Hence, $\texttt{def}(s^*, s', [t, t_{i,j+1} + 1)) > (\alpha - \beta)C$, that is, $\texttt{hold}(\sigma^*, \overrightarrow{\mathcal{P}_{i,j}}) > (\alpha - \beta)C$, and the claim follows.

The rest of the proof is identical to Lemma 3. □

Theorem 4 *The competitive ratio of* DTrack−B *is bounded as follows:*

$$\begin{array}{lll} r(\texttt{DTrack}) < 1 + (\texttt{k}-1)\alpha + \texttt{k} & \alpha \geq 1 \text{ and } \beta \leq \alpha - 1 & (1) \\ r(\texttt{DTrack}) < \texttt{k}(1 + \frac{1}{\alpha}) & \alpha \leq 1 \text{ and } \beta \leq 0 & (2) \\ r(\texttt{DTrack}) < 1 + \frac{(\texttt{k}-1)\alpha+\texttt{k}}{\alpha-\beta} & \max(0, \alpha-1) \leq \beta \leq \alpha & (3) \end{array}$$

Proof : Consider the local ratio between the costs incurred by DTrack−RR and OPT during a single OPT-round $[\tau_i, \tau_{i+1})$. Similarly to the proof of Theorem 4, we derive

$$\frac{\texttt{cost}(\sigma, [\tau_i, \tau_{i+1}))}{\texttt{cost}(\sigma^*, [\tau_i, \tau_{i+1}))} < 1 + \frac{p((k-1)\alpha + k)}{1 + (p-1)\min(\alpha, \alpha - \beta)}.$$

We denote $\varrho(\alpha, \beta, p) \triangleq \frac{\texttt{cost}(\sigma,[\tau_i,\tau_{i+1}))}{\texttt{cost}(\sigma^*,[\tau_i,\tau_{i+1}))}$.

If $\min(\alpha, \alpha - \beta) \geq 1$, (i.e., $\alpha \geq 1$ and $\alpha - \beta \geq 1$), then the derivative $\frac{\partial \varrho}{\partial p}$ is non-negative, and hence,

$$\sup_{1 \leq p < \infty} \varrho(\alpha, \beta, p) = \varrho(\alpha, \beta, 1) = 1 + (k-1)\alpha + k \quad \text{if } \alpha \geq 1 \text{ and } \beta \leq \alpha - 1. \tag{1}$$

If $\min(\alpha, \alpha - \beta) \leq 1$, then $\frac{\partial \varrho}{\partial p}$ is non-positive, and hence,

$$\sup_{1 \leq p < \infty} \varrho(\alpha, \beta, p) = \lim_{p \to \infty} \varrho(\alpha, \beta, p) = 1 + \frac{(k-1)\alpha + k}{\min(\alpha, \alpha - \beta)}.$$

Consider the case when $\min(\alpha, \alpha - \beta) = \alpha$, i.e., $\beta \leq 0$. Combining this with $\alpha \leq 1$, we get:

$$\sup_{1 \leq p < \infty} \varrho(\alpha, \beta, p) = 1 + \frac{(k-1)\alpha + k}{\alpha} = k(1 + \frac{1}{\alpha}) \quad \text{if } \alpha \leq 1 \text{ and } \beta \leq 0. \tag{2}$$

Consider the case when $\min(\alpha, \alpha - \beta) = \alpha - \beta$, i.e., $\beta \geq 0$. Combining this with $\alpha - \beta \leq 1$ and $\beta \leq \alpha$ (by definition), we get:

$$\sup_{1 \leq p < \infty} \varrho(\alpha, \beta, p) = 1 + \frac{(k-1)\alpha + k}{\alpha - \beta} \leq \quad \text{if } \max(0, \alpha - 1) \leq \beta \leq \alpha \tag{3}$$

□

A.4 Non-Competitiveness of Opportunistic Algorithms

In this section, we show that the opportunistic versions of DTrack are not competitive, that is, the worst-case competitive ratio depends on C, rather than on the problem size k.

Theorem 5 *The competitive ratio of* DTrack−B *with* $\alpha = \beta$ *is* $\Omega(C)$.

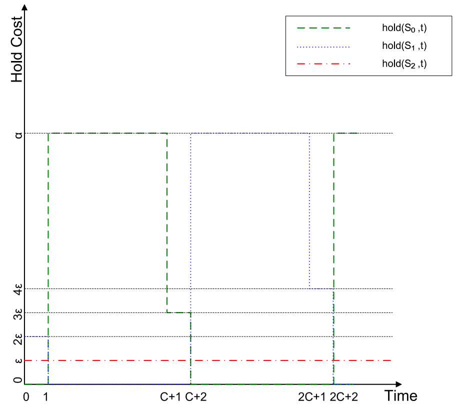


Figure 10: **An example of hold costs for which `DTrack−B` with $\alpha = \beta$ is $\Omega(C)$-competitive.**

Proof : Assume wlog that C is a positive integer. Let ε be a small number s.t. $0 < \varepsilon < \frac{\alpha}{C}$. Consider three servers s_0, s_1 and s_2, which have a constant setup cost C. Let $\texttt{hold}(s_2, t) = \varepsilon$ for all t, whereas $\texttt{hold}(s_0, t)$ and $\texttt{hold}(s_1, t)$ are defined as follows for integer values of $0 \leq i < \lceil \frac{C}{2} \rceil$:

$$\texttt{hold}(s_0, t) = \begin{cases} (2i+3)\varepsilon & t = (2i+1)(C+1) \\ \alpha & 2i(C+1) < t < (2i+1)(C+1) \\ 0 & \text{otherwise} \end{cases}$$

and

$$\texttt{hold}(s_1, t) = \begin{cases} (2i+2)\varepsilon & t = 2i(C+1) \\ \alpha & (2i+1)(C+1) < t < (2i+2)(C+1) \\ 0 & \text{otherwise} \end{cases}$$

The hold costs during the interval $[0..2C+2]$ are depicted in Figure 10.

Consider a run of `DTrack−B` during the interval $[0 \ldots (C-1)(C+1)]$. Note that by definition, $\texttt{hold}(s_0, t) \leq \alpha$ and $\texttt{hold}(s_1, t) \leq \alpha$ for all t during this interval. At time $t = 0$, `DTrack−B` chooses s_0 because it offers the minimal hold cost. It is easy to show by induction that the algorithm assigns s_0 at times $2i(C+1)$, and s_1 at times $(2i+1)(C+1)$, for $0 \leq i < \lceil \frac{C}{2} \rceil$. Consider first $t = (2i+1)(C+1) \leq C(C+1)$. The algorithm transitioned to s_0 at $2i(C+1)$ by the induction hypothesis. Therefore,

$$\texttt{def}(s_0, s_1, [2i(C+1), (2i+1)(C+1))) = -(2i+2)\varepsilon + \alpha C + ((2i+1)+2)\varepsilon = \alpha C + \varepsilon > \alpha C,$$

whereas

$$\texttt{def}(s_0, s_2, [2i(C+1), (2i+1)(C+1))) = -\varepsilon + C(\alpha - \varepsilon) + ((2i+1)+2-1)\varepsilon = \alpha C - \varepsilon(C-(2i+1)) \leq \alpha C.$$

Therefore, the next transition happens at $t = (2i+1)(C+1)$, and s_2 is chosen as the one that achieves the largest deficit. The case $t = (2i+2)(C+1)$ is proved analogously.

`DTrack−B` transitions C times during the interval, and hence, the total setup cost is C^2. The total hold cost exceeds αC^2 since between every two transitions the algorithm incurs above αC. Hence, the total cost during the interval exceeds $(\alpha+1)C^2$. Besides, `OPT` would select s_2 at $t = 0$ and never change its assignment, therefore paying a total setup cost of C and the total hold cost of $\varepsilon(C-1)(C+1) < \alpha C$. Hence, the total cost incurred by it is less than $(\alpha+1)C$, and the competitive ratio is $\Omega(C)$. □

Theorem 6 *The competitive ratio of* `DTrack−F` *is* $\Omega(C)$.

Proof : The same proof works here, because upon every transition, the server with the zero hold cost is selected. That is, `DTrack−F` bounces between s_0 and s_1, while `OPT` would select s_2 and never transition from it. □

References

[1] I.F. Akyldiz, X. Wang, and W. Wang. Wireless Mesh Networks: a Survey. *Computer Networks Journal (Elsevier)*, March 2005.

[2] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-Based Load Management in Federated Distributed Systems. *1st Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[3] Y. Bejerano, I. Cidon, and J. Naor. Dynamic Session Management for Static and Mobile Users: a Competitive On-Line Algorithmic Approach (Part II). *Accepted to ACM Transactions on Networking.*

[4] S. Bespamyatnikh, B.Bhattacharya, D.Kirkpatrick, and M.Segal. Mobile Facility Location. *ACM Workshop on Foundations of Mobile Computing (DIALM)*, 2004.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[7] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. *ACM MOBICOM*, September 2004.

[8] S. Jamin, C. Jin, D. Raz, and Y. Shavitt. Constrained Mirror Placement on the Internet. *IEEE INFOCOM*, 2001.

[9] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. *Proceedings of HotNets*, 2003.

[10] N. Lavi, I. Cidon, and I.Keidar. MaGMA: Mobility and Group Management Architecture for Real-Time Collaborative Applications. *Accepted to Wiley J. on Wireless Communication and Mobile Computing (WCMC).*

[11] K.-W. Lee, B.-J. Ko, and S. Calo. Adaptive Server Selection for Large Scale Interactive Online Games. *ACM Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2004.

[12] A. Meyerson. Online Facility Location. *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

[13] P. B. Mirchandani and R. L. Francis. *Discrete Location Theory.* John Wiley & Sons Inc., 1990.

[14] L. Qiu, V.N.Padmanabham, and G.M.Voelker. On Placement of Web Server Replicas. *IEEE INFOCOM*, 2001.

[15] J. Yoon, M. Liu, and B. Noble. Sound Mobility Models. *ACM MOBICOM*, 2003.