Nomadic Service Assignment

Edward Bortnikov, Israel Cidon, and Idit Keidar

Abstract—We consider the problem of dynamically assigning application sessions of mobile users or user groups to service points. Such assignments must balance the trade-off 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 heuristics that achieve near-optimal 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.

Index Terms—Mobile communication systems, online computation, distributed networks.

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], [12], [14]. 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. The mesh access protocol typically routes the traffic of each mobile node through a single access gateway. As the node travels away from its original location, the network delay between it and the gateway grows, and the protocol can reroute the traffic through a different gateway to improve the QoS. For example, a greedy protocol would always route the traffic via the closest gateway. However, this optimization is not always adequate for highly mobile users, which suffer from QoS degradation caused by frequent handoffs. Intelligent nomadic service assignment algorithms can mitigate the trade-off between access delay and session interruptions.

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. Thus, there is a trade-off between the cost of assignment to a suboptimal server (e.g., increased delay) and the cost of state transfer incurred upon the reassignment.

In this paper, we study the problem of optimizing the dynamic assignment of sessions to service points. Such a service assignment should balance the trade-off between connecting sessions to the closest servers at all times and minimizing the number of session migrations. We capture this trade-off 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. For simplicity, 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. For example, in a mobile WMN, connection transfers are done through wired infrastructure of predictable performance. In this context, the setup cost is fixed, since it does not depend on the
location of the source and target gateways. The hold costs, which capture user-gateway distances, are variable.

The nomadic service assignment optimization problem is to find 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. We first handle the generic nomadic service assignment problem and then examine it more closely in two specific case studies pertaining to specific example domains.

We formally define the problem in Section 3. Then, in Section 4, 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 5, we study nomadic service assignment as an online optimization problem. A common metric for an online algorithm 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 $(2 + \alpha)k$, where $\alpha$ 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 worldwide. 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 6 and 7 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 percent above the optimum in the average case in the WMN (for a widely accepted random waypoint mobility model, e.g., [22]), and at most 20 percent above optimal in the groupware service (for uniformly distributed users with a Poisson arrival process). More importantly, this ratio, as well as the total cost, remains almost constant as the problem size scales.

There is a trade-off 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 6.1, 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 look-ahead window is quite small (the node’s next target), both motion-aware algorithms yield significant cost improvements. Their costs are typically within 10 percent above the optimum and exhibit perfect scalability.

2 RELATED WORK

Handoff optimizations in mobile systems have been extensively studied since the early 1990s, mostly in the context of cellular networks with the advent of the GSM standard [11], [19], [21]. This research targeted increasing network capacity as the primary goal. Handoffs in cellular systems are driven by physical metrics, like signal strength and transmission power, and are handled at the link layer. They cannot be avoided when user location changes significantly, and optimizing their cost is a secondary design goal (e.g., [9]). Our work is fundamentally different because we consider the network layer and above. In this context, handoffs are optional, and they improve the QoS in the long run, but their cost is substantial. For example, migrating a host connection between two WMN gateways can affect packet delivery order and temporarily degrade the TCP performance.

Initial mesh networking research mostly focused on problems that are specific to fixed wireless, e.g., defining routing metrics [12], exploiting the broadcast nature of the medium [6], and harnessing multiple radio interfaces through smart cross-layer design [2]. More recently, Amir et al. presented the design and implementation of a prototype WMN with mobility support [3]. The algorithms presented in this paper can be integrated into such a system for handoff decisions. Lavi et al. [15] proposed employing an overlay service network for supporting groupware in mobile networks. Their architecture suggested associating every mobile user with the closest server and efficiently maintaining the group membership information between multiple servers. In contrast with this approach, we focus on applications (including possibly groupware) that associate a session with a single server.
The problem of dynamic session management was studied in the context of routing virtual circuits in mobile ATM networks [4] 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. This model allowed reusing part of the links after the rerouting, thus allowing for lower total costs than in our model where no reuse is possible. Indeed, their algorithms exhibit better competitive behavior than the best possible for nomadic service assignment.

Nomadic service assignment is closely related to the classical metrical task system (MTS) problem [8]. In this context, there is a set of $k$ states and a matrix of interstate transition costs (the cost of a self-transition is zero). A schedule for a sequence of tasks is a sequence of states in which these tasks are processed. The cost of a schedule is the sum of all task processing (hold) and transition (setup) costs. For symmetric cost matrices subject to the triangle inequality, there is a deterministic online algorithm with a competitive ratio of $2k - 1$, and this bound is tight. Nomadic service assignment closely resembles a special case of this problem with uniform transition costs, except that, in our problem, the initial assignment always incurs a cost. However, the online MTS algorithm [8] makes use of the entire history of setup and hold costs until the scheduling decision, which makes it impractical to implement. We use a very different algorithmic technique, which requires $O(k)$ operations per decision regardless of the history length. In a specific setting of a WMN with mobile users in which the hold costs are defined as user-gateway distances, the computation overhead of our algorithm can be further reduced by an order of magnitude through the use of spatial data structures.

Optimal center location for a group of users is an instance of the well-studied facility location problem [18], 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 [17] and was used for various applications, including optimizing the delivery of Web content in CDN’s [13], [20], maintenance of mobile centers in ad hoc networks [5], and adaptive server selection in online games [16]. 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.

3 System Model

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

There are two types of nonnegative costs 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 $\text{setup}(s, t)$ and the hold cost by $\text{hold}(s, t)$.

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

$$T(\sigma, I) = \{t | t \in I \land \sigma(t) \neq \sigma(t - 1)\}.$$

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

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

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

a total setup cost

$$\text{setup}(\sigma, [t_1, t_2)) = \sum_{t \in T(\sigma, [t_1, t_2))} \text{setup}(\sigma(t), t),$$

and a total overall cost

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

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

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 positive setup costs.

4 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 $\sigma^*$. Let $\sigma_{s, t}^* : [t, T) \rightarrow S$ 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

$$\text{cost}(\sigma_{s, t}^*, [t + 1, T)) = \text{cost}(\sigma_{s', t + 1}^*, [t + 1, T)).$$

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 $\text{setup}(s', t + 1)$ does not contribute to $\text{cost}(\sigma_{s, t}^*, [t, T))$.

The problem can be represented as a layered directed acyclic graph. Node $i$ in layer $t$ stands for $\sigma_{s, t}^*$ for $1 \leq i \leq k$, $0 \leq t \leq T$. There is an edge between every pair of nodes $(i, t)$ and $(j, t + 1)$, which represents a possible transition from $s_i$ to $s_j$ at time $t$. The cost of this edge is $\text{hold}(s_i, t + 1)$ if $i = j$ and $\text{hold}(s_j, t + 1) + \text{setup}(s_j, t + 1)$ otherwise. The optimal solution’s cost is the weight of the shortest path in the graph. While this weight can be computed in linear time
in the number of edges, i.e., $O(k^2T)$, the time complexity can be optimized to $O(kT)$ by exploiting the optimal solution’s structure, as we now explain.

We define the tail contribution function for $t < T$ as follows:

$$
tail(s, s', [t, T]) = \begin{cases} 
\text{cost}(\sigma_{s,t}^*, [t, T]) - \text{setup}(s, t) & \text{if } s = s' \\
\text{cost}(\sigma_{s,t}^*, [t, T]) & \text{otherwise.}
\end{cases}
$$

Then, $\text{cost}(\sigma_{s,t}^*, [t, T])$ for $t < T$ can be expressed as

$$
\text{cost}(\sigma_{s,t}^*, [t, T]) = \text{setup}(s, t) + \text{hold}(s, t) + \min_{s' \in S} \text{tail}(s, s', [t + 1, T]).
$$

We define $\text{tail}(s, s', [t, T])$ as $\text{cost}(\sigma_{s,t}^*, [T, T]) = 0$. For $t < T$, we get

$$
\text{cost}(\sigma_{s,t}^*, [t, T]) = \text{setup}(s, t) + \text{hold}(s, t) + \min_{s' \in S} \left( \min_{0 \leq i \leq k} \text{cost}(\sigma_{s',t+1}^*, [t + 1, T]), \text{cost}(\sigma_{s,t+1}^*, [t + 1, T]) \right) - \text{setup}(s, t + 1)).
$$

An optimal solution can be computed through dynamic programming [10] using the above recurrence. The algorithm employs a two-dimensional table $\text{Table}[1..k,0..T]$, where an entry $\text{Table}[s,t]$ holds the value of $\text{cost}(\sigma_{s,t}^*, [t, T])$ 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

$$
\min_{s' \in S} \text{cost}(\sigma_{s',t}^*, [t, T]) = \min_{0 \leq i \leq k} \text{Table}[s,t]
$$
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

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

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 precomputation of the previous column’s minimum cost and, therefore, the algorithm’s time complexity is $O(kT)$. The space complexity is also $O(kT)$ — the table’s size.

## 5 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 [7]. 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, $\text{setup}(s, t) = C$ for all $s$ and $t$, whereas the hold costs are dynamic. We denote the schedule produced by the optimal algorithm $\text{OPT}$ as $\sigma^*$ and the schedule produced by an online algorithm $\text{ALG}$ as $\sigma$.

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

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

The rest of this section is structured as follows: In Section 5.1, we show that no deterministic online algorithm can achieve a competitive ratio better than $k$. In Section 5.2, 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 5.3, we present a simple and efficient algorithm called $CTrack$ (cost tracker), which yields a competitive ratio of $(2 + \alpha)k$ for a certain parameter choice, assuming that a server’s per-slot hold cost never exceeds $\alpha C$. The competitive version of $CTrack$, called $CTrack-RR$, probes the cost of only one server every slot. In Section 5.4, we present opportunistic versions of these algorithms, called $CTrack-F$, $DTrack-F$, and $DTrack-B$, which are not competitive but greatly improve the cost in the average case and achieve good scalability.

### 5.1 A Lower Bound of $k$ on the Competitive Ratio

**Theorem 1.** No deterministic server assignment algorithm can achieve a competitive ratio of less than $k$.

**Proof.** Consider $k$ symmetric servers that offer the same setup cost $C > 0$ and a zero hold cost each at $t = 0$, that is, $\text{hold}(s, 0) = 0$. Consider the following simple adversary strategy against any deterministic algorithm $\text{ALG}$. When $\text{ALG}$ connects to $s_i$ at time $t$, set $\text{hold}(s_i, t+1) = 1$. When $\text{ALG}$ disconnects from the server at time $t'$, set $\text{hold}(s_i, t'+1) = 0$. 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 $\text{ALG}$ has visited every server exactly once, let $s^*$ be its last assignment. Otherwise, there exists a server $s^*$ that has never been picked by $\text{ALG}$. The best offline algorithm, $\text{OPT}$, assigns the session to server $s^*$ at time 0 and never changes the assignment.

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

### 5.2 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 $\text{nextchoice}()$, 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 $\text{nextchoice}()$ policy. Its pseudocode appears in Fig. 1.
1: Initialization:
2: \( t \leftarrow 0 \)
3: \( c \leftarrow i \) s.t. \( \text{hold}(s_i, 0) = \min_{s \in S} \text{hold}(s, 0) \)
4: \( \text{reset()} \)

5: Every time slot do
6: \( \text{update()} \)
7: if (\( \text{Def}[s] > \alpha C \)) for some \( s \in \text{Leaders} \) then
8: \( \text{nextchoice()} \)
9: \( \text{reset()} \)
10: \( t \leftarrow t + 1 \)

11: procedure \( \text{reset()} \)
12: \( \tau \leftarrow t \)
13: \( \text{Leaders} \leftarrow 0 \)
14: for all \( s \neq s_e \) do
15: \( \text{Def}[s] \leftarrow 0 \)
16: \( \text{Leaders} \leftarrow \text{Leaders} \cup \{s\} \)

17: procedure \( \text{update()} \)
18: for all \( s \) s.t. \( s \notin \text{Leaders} \wedge \text{hold}(s_e, t) > \text{hold}(s, t) \) do
19: \( \text{Def}[s] \leftarrow 0 \)
20: \( \text{Leaders} \leftarrow \text{Leaders} \cup \{s\} \)
21: for all \( s \in \text{Leaders} \) do
22: \( \text{Def}[s] \leftarrow \text{Def}[s] + \text{hold}(s_e, t) - \text{hold}(s, t) \)
23: if (\( \text{Def}[s] < 0 \)) then
24: \( \text{Leaders} \leftarrow \text{Leaders} \setminus \{s\} \)

25: procedure nextchoice() /*RR version*/
26: repeat
27: \( c \leftarrow (c + 1) \mod k \)
28: until \( \text{hold}(s_e, t) - \min_{s \in S} \text{hold}(s, t) \leq \alpha C \)

Fig. 1. \( \text{DTrack-RR} \)—an online algorithm for server assignment.

We begin with some definitions. The \textit{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 \([t', t]\):

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

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

The algorithm’s code maintains the following variables: \( t \) is the current time, \( \tau \) is the last transition’s time, \( c \) is the current assignment’s id, \( \text{Leaders} \) is the set of current leaders’ ids, and \( \text{Def} \) is the vector of deficit values between \( s_c \) and the other servers. The algorithm maintains that at time \( t \), \( \text{Def}[s] = \text{def}(s_c, s, [\tau, t + 1]) \).

\text{DTrack} maintains an invariant that the deficit between \( s_c \) and any other server \( s \) never exceeds \( \alpha C \). Initially, \text{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 cumulative deficit value becomes negative. Since the hold costs are published at the beginning of each time slot, \text{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 \( \alpha C \)), the algorithm changes its assignment. Due to the lookahead mechanism, the \text{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 \text{DTrack} we consider now, termed \text{DTrack-RR}, \text{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 \( \alpha C \).

The intuition behind \text{DTrack} is that the current server must be provably bad (costing \( \alpha C \) more than the best) in order to change the choice, and the next server must also not be provably bad (not costing \( \alpha C \) more than any other server). When instantiated with \( \alpha = 0 \) (this algorithm is termed \text{Greedy}), \text{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 \text{DTrack-RR} and get the following result:

**Theorem 2.** The competitive ratio of \text{DTrack-RR} is bounded as follows:

\[
r(\text{DTrack-RR}) < k \left( 1 + \frac{1}{\alpha} \right) \quad \alpha \leq 1
\]

\[
r(\text{DTrack-RR}) < 1 + (k-1)\alpha + k \quad \alpha \geq 1.
\]

**Corollary 1.** For \( \alpha = 1 \), \text{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 \( \sigma \) by \text{DTrack-RR} that \textit{overtakes} (that is, either leaves or skips) every server while the optimal schedule \( \sigma^* \) does not change its assignment \( s^* \), then \( \sigma \) overtake \( s^* \) exactly once. This overtake implies that the total hold cost incurred by \( \sigma \) during the interval exceeds \( \alpha C \). The total hold cost incurred by \( \sigma \) exceeds the one incurred by \( \sigma^* \) by at most \((k-1)\alpha C \). The subtle point in this proof is that the deficit bookkeeping because, upon transition, the hold cost lookahead affects the assignment but does not contribute to the total hold cost. The total setup cost incurred by \( \sigma \) during this period is at most \( kC \), whereas \( \sigma^* \) pays \( C \) upon the assignment to \( s^* \). A careful analysis of the worst-case ratio between the total costs concludes the proof.

### 5.3 \text{CTrack—An Efficient Online Algorithm}

At each slot, \text{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.

We now present a simple online algorithm \text{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. \text{CTrack} is also parameterized by a factor \( \alpha \) and a subroutine \text{nextchoice}(). Initially, it
assigns the server with the minimal hold cost. The assignment changes when the total hold cost since the last transition exceeds $\alpha C$ (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 \( \textsc{CTrack-RR} \), the round-robin version of \( \textsc{CTrack} \), and get the following result:

**Theorem 3.** If \( \text{hold}(s,t) \leq \alpha C \) for all \( s \) and \( t \), then \( r(\text{\textsc{CTrack-RR}}) < (2 + \alpha)k \) for \( \alpha = 1 \).

### 5.4 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 2\( k \) 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 \( \textsc{CTrack} \) and \( \textsc{DTrack} \), in which \texttt{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 \( \textsc{DTrack-F} \) and \( \textsc{CTrack-F} \), \texttt{nextchoice()} picks the server with the current minimal hold cost. The backward heuristic \( \textsc{DTrack-B} \) augments \( \textsc{DTrack-RR} \)’s selection policy with the following rule: The deficit between the next choice and the previous assignment is greater than \( \beta C \) for some \( -\infty < \beta \leq \alpha \). 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 \( \textsc{DTrack-RR} \). For \( \beta = 0 \), \( \textsc{DTrack-B} \) 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 \( \textsc{DTrack-B} \)’s worst-case behavior (the proof appears in Section A.3):

**Theorem 4.** The competitive ratio of \( \textsc{DTrack-B} \) is bounded as follows:

\[
\begin{align*}
r(\text{\textsc{DTrack}}) &< k \left(1 + \frac{1}{\alpha}\right) & \alpha \leq 1 \text{ and } \beta \leq 0 \\
r(\text{\textsc{DTrack}}) &< 1 + k - \alpha \alpha + k & \alpha \geq 1 \text{ and } \beta \leq \alpha - 1 \\
r(\text{\textsc{DTrack}}) &< 1 + \frac{(k - 1)\alpha + k}{\alpha - \beta} & \max(0, \alpha - 1) \leq \beta \leq \alpha.
\end{align*}
\]

**Corollary 2.** For \( \alpha = 1 \) and \( \beta \leq 0 \), \( \textsc{DTrack-B} \) achieves a competitive ratio of \( 2k \).

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

In this section, we study nomadic service assignment in an urban WMN environment. The results of the optimal algorithm \textsc{Opt} are used as a comparison baseline. For each algorithm \textsc{ALG}, we measure its cost as well as performance ratio, which is the average ratio between the total costs incurred by \textsc{ALG} and \textsc{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 1,000 m \( \times \) 1,000 m grid is 100; that is, a single router spans an average area of 100 m \( \times \) 100 m. A mobile node moves using the random waypoint mobility model [22]. 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.

We assume that the wireless infrastructure is the main bottleneck, whereas the gateway resources are abundant and, hence, the end-user QoS is not affected by the congestion among multiple connections. The hold cost between mobile node \( n \) and router \( r \) is defined as \( d(n,r) - d_{	ext{router}}(n,r) = \sqrt{d(n,r)} \), 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 1,000 m \( \times \) 1,000 m to 5,000 m \( \times \) 5,000 m and correspondingly increase the number of routers from 100 to 2,500, keeping the router density fixed. We study the performance of different versions of \( \textsc{CTrack} \) and \( \textsc{DTrack} \) with different selections of \( \alpha, \beta \), and \texttt{nextchoice()}.

Our first goal is to study the performance of \( \textsc{CTrack-RR} \) and \( \textsc{DTrack-RR} \) with \( \alpha = 1 \), which have the best proven worst-case ratios. Fig. 2 shows that both algorithms scale poorly with the network size (their costs grow approximately as \( \sqrt{k} \), whereas \textsc{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 \( O(\sqrt{k}) \).

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

Figs. 4a and 4b depict the results of simulating the opportunistic algorithms \textbf{Greedy}, \textbf{CTrack-F}, \textbf{DTrack-F}, and \textbf{DTrack-B} with \( \alpha = 1 \) and \( \beta = 1 \). The performance curves of \textbf{CTrack-F} and \textbf{DTrack-F} are almost indistinguishable. The algorithms' performance ratios remain constant as the problem scales—around 50 percent above the optimum. The total cost per second also remains constant since \textbf{OPT} itself is very scalable. \textbf{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, \textbf{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 \textbf{DTrack-F} consistently produces better results than \textbf{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).

Fig. 4c depicts the results of the same experiment with an average simulated speed 25 m/sec (90 km/hour). In this setting, \textbf{DTrack-F} starts producing a consistently lower total cost (by 5-6 percent) than \textbf{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 7). As expected, \textbf{Greedy} performs worse at higher speeds (above five times the optimum).

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

\textbf{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 structures that support efficient nearest neighbor queries in a multidimensional space like KD-trees or R-trees [18]. Fig. 6 depicts the percentage of hold costs that need to be accessed by \textbf{DTrack-F} and \textbf{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.

### 6.1 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 \textbf{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 \( \text{OPT} \) 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 \( \text{OPT} \) with the fixed first assignment that is identical to the current one (i.e., no setup cost is incurred for it) or letting \( \text{OPT} \) 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 estimated 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.

Fig. 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 \( \text{CTrack-F} \) and \( \text{DTrack-F} \). Their performance ratios are less than 10 percent and 18 percent 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 \( \text{OPT} \) because their lookahead window grows as the grid scales up.

We also evaluated DirectionAware’s capability to handle inaccurate predictions by supplying it with direction estimates that are normally distributed around the real direction with variance \( \varepsilon \). The values of \( \varepsilon \) ranged from 0 degrees (exact prediction) to 30 degrees (Fig. 7). As expected, the algorithm’s performance ratio grows with \( \varepsilon \). However, this growth is limited by 25 percent above the optimum, i.e., only 7 percent above the algorithm with a perfect direction forecast. Therefore, DirectionAware is quite tolerant to moderately inaccurate direction estimates.

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

7 CASE STUDY: WIDE-AREA CHATROOM SERVICE

The second environment studied is an Internet-scale groupware application service [15], [16], 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 \( \lambda \), and the membership lifetime is distributed exponentially with a mean of \( T \) (that is, the average number of users in a group is \( \lambda T \)). 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 6 (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 2,500 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. Fig. 8a depicts the simulation results. Both versions of DTrack are within 15-20 percent 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 (Fig. 4c), but it is more significant since the hold cost changes are faster.

In the second experiment, depicted in Fig. 8b, we scale the average group size up from three to 75 (a large-scale conference) by increasing both $\lambda$ and $T$. The network size is not changed. Both versions of DTrack exhibit a performance ratio of under 5 percent above the optimum for groups with more than 10 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.

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 Fig. 8c show that, when the number of servers grows from 400 to 2,500 and the number of users grows from 12 to 75, the performance ratios of both versions of DTrack remain constant at less than 5 percent above the optimum, whereas the performance ratio of CTrack-F also remains constant but exceeds the optimum by 30 percent.

8 CONCLUSION

In this paper, we have studied a 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 near-optimal performance and scale well with the network size.

APPENDIX 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 \( \alpha \) for which the best competitive ratio is obtained.

Claim 1. Let \( t \) be a time and \( s \) a server. Let \( \tau \) be the time of the latest transition before \( t + 1 \). DTrack-RR maintains that

\[
\text{Def}(s, s, [\tau, t + 1]) = \alpha C.
\]

Proof. Immediate from the code (Lines 14-15 stand for the initialization upon assignment and Lines 18-24 stand for the maintenance between assignments). \( \square \)

Lemma 1. Let \( t \) be a time and \( s \) be a server. Let \( \tau \) be the time of the latest transition before \( t + 1 \). Then,

\[
\text{Def}(\sigma(\tau), s, [\tau, t + 1]) \leq \alpha C.
\]

Proof. By induction on \( t \). For \( t = 0 \), the claim holds because the server with the minimal hold cost is selected (Line 3). For \( t > 0 \), if there is no transition at \( t \), then the invariant is maintained by the algorithm’s code (Line 7). Assume that a transition occurs at time \( t \), i.e., \( \tau = t \). By the induction hypothesis, \( \text{Def}(\sigma(\tau'), s, [\tau', t]) \leq \alpha C \), where \( \tau' \) is the previous transition time. However, since a transition happened at \( t \), then, for some \( s \), \( \text{Def}(\sigma(\tau'), s, [\tau', t + 1]) > \alpha C \). Hence, there exists some server \( s' \) such that \( \text{hold}(s, t) < \text{hold}(\sigma(\tau'), t) \), that is, \( \text{hold}(\sigma(\tau'), t) \) is not the minimal hold cost at time \( t \). Therefore, some identity \( s \neq \sigma(\tau') \) can be found such that \( \text{Def}(s, s', [t, t + 1]) \leq \alpha C \) for all \( s' \) (Line 28), e.g., the server with the minimal hold cost at \( t \) satisfies this requirement. \( \square \)

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 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 \((s + 1) \mod k\). We say that \( \sigma \) 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 \( \sigma(t - 1) = s \) or \( s \) is skipped at \( t \).

We now consider a DTrack-RR-round and an ALG-round of an arbitrary algorithm ALG. We analyze the competitive ratio of DTrack-RR for different values of \( \alpha \) by comparing the cost it incurs with the cost incurred by ALG during a single ALG-round \([\tau_i, \tau_{i+1})\) and then generalizing for the whole run. We denote ALG’s schedule by \( \sigma' \) and ALG’s assignment during this ALG-round by \( s' \) (if ALG is OPT, the notations are \( \sigma' \) and \( s' \), respectively).

We define two partitions of the interval \([\tau_i, \tau_{i+1})\) into subintervals. The first one partitions the interval to phases \( \{P_{ij} = [t_i, t_{ij+1})\} \), defined as follows: The first phase starts at \( \tau_i \). A phase completes at the earlier between the time \( \sigma \) overtake \( s' \) and \( \tau_{i+1} \). The second partition is to shifted phases \( \{P_{ij}^{\prime}\} \), defined as follows: The first shifted phase starts at \( \tau_i \). A shifted phase completes at the earlier between one slot after the completion of the corresponding phase and \( \tau_{i+1} \).

Fig. 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 \( s_8 \) 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 ALG-round \([\tau_i, \tau_{i+1})\) with \( p \) phases produced by DTrack-RR. Then,

\[
\text{cost}(\sigma, [\tau_i, \tau_{i+1}) \leq \text{hold}(\sigma', [\tau_i, \tau_{i+1}) + pC(k + (k - 1)\alpha).
\]

Proof. Consider a DTrack-RR-round \([t, t') \subseteq P_{ij} \) and denote \( s = \sigma(t) \).

If \( s = s' \), then \( \text{hold}(\sigma', [t, t')) = \text{hold}(\sigma, [t, t')) \). Otherwise, by the definition of \( \text{def} \),

\[
\text{hold}(\sigma, [t, t')) - \text{hold}(\sigma', [t, t')) \leq \text{def}(s, s', [t, t')).
\]

By Lemma 1, \( \text{def}(s, s', [t, t')) \leq \alpha C \). Therefore,

\[
\text{hold}(\sigma, [t, t')) - \text{hold}(\sigma', [t, t')) \leq \alpha C.
\]
There are at most \( k - 1 \) rounds during \( P_{ij} \) in which the assignment is different from \( s' \) and, hence,
\[
\text{hold}(\sigma, P_{ij}) - \text{hold}(\sigma', P_{ij}) \leq (k - 1) \alpha C.
\]

DTrack-RR performs at most \( k \) transitions during \( P_{ij} \), paying at most \( kC \) for setup. Therefore,
\[
\text{cost}(\sigma, P_{ij}) \leq \text{hold}(\sigma', P_{ij}) + (k - 1) \alpha C + kC.
\]

\( \{P_{ij}\} \) is a partition of \( [\tau_i, \tau_{i+1}] \) and, hence,
\[
\text{cost}(\sigma, [\tau_i, \tau_{i+1}]) = \sum_{j=1}^{n} \text{cost}(\sigma, P_{ij}) \leq \sum_{j=1}^{n} \text{hold}(\sigma', P_{ij}) + pC(k + (k - 1)\alpha) \\
= \text{hold}(\sigma', [\tau_i, \tau_{i+1}]) + pC(k + (k - 1)\alpha).
\]

\[\Box\]

Lemma 3. Consider an ALG-round \( [\tau_i, \tau_{i+1}] \) with \( p \) phases produced by DTrack-RR, such that either \( \sigma(\tau_i - 1) \neq \sigma'(\tau_i) \) or \( \sigma(\tau_i) \neq \sigma'(\tau_i) \). Then, \( \text{hold}(\sigma', [\tau_i, \tau_{i+1}]) \geq (p - 1)\alpha C \).

Proof. If \( p = 1 \), the claim trivially holds because the hold costs are nonnegative.

Otherwise, consider a phase \( P_{ij} \) such that \( j < p \). This phase ends at \( t_{i+j+1} \), which is strictly smaller than \( \tau_{i+1} \). We first prove a claim that \( \text{hold}(\sigma', P_{ij}) > \alpha C \). Consider DTrack-RR’s assignment \( s \) during the last DTrack-RR-round \( [t, t_{i+j+1}] \) in \( P_{ij} \); that is, \( s = \sigma(t) \) and \( s \) overtake \( s' \) at time \( t_{i+j+1} \). By definition, \( P_{ij} \) ends at time \( t_{i+j+1} \). Consider two possible cases:

1. If \( s \neq s' \), then the algorithm considers \( s' \) upon the transition from \( s \) at \( t_{i+j+1} \) and does not select it because there exists a server \( \tilde{s} \) such that
\[
\text{hold}(s', t_{i+j+1}) - \text{hold}(\tilde{s}, t_{i+j+1}) > \alpha C;
\]

hence, \( \text{hold}(s', t_{i+j+1}) > \alpha C \). By the definition of a shifted phase, \( [t_{i+j+1}, t_{i+j+1} + 1] \subseteq P_{ij} \). It follows that \( \text{hold}(\sigma', P_{ij}) > \alpha C \) and the claim holds.

2. Otherwise, \( s = s' \). Since the algorithm transitions from \( s' \) at time \( t_{i+j+1} \), there exists a such that \( \text{def}(s', \tilde{s}, t_{i+j+1} + 1) > \alpha C \), that is,
\[
\text{hold}(\sigma', [t, t_{i+j+1} + 1]) > \alpha C.
\]

Assume that \( P_{ij} \) is the first phase in \( [\tau_i, \tau_{i+1}] \). Since either \( \sigma(\tau_i - 1) \neq \sigma'(\tau_i) \) or \( \sigma(\tau_i) \neq \sigma'(\tau_i) \), DTrack-RR’s assignment to \( s' \) did not happen before \( \tau_i \), i.e.,
\( t_{i+j+1} \). Hence, \( [t, t_{i+j+1} + 1] \subseteq P_{ij} \) by the definition of a shifted phase. Otherwise, consider the preceding phase \( P_{ij-1} \). By definition, \( \sigma \) overtake \( s' \) at time \( t_{i+j} \). In particular, \( \sigma(t_{i+j}) \neq s' \). Since at least one time slot is spent at every assignment, \( \sigma \) transitions to \( s' \) at time \( t_{i+j} < t < t_{i+j+1} \), that is, \( [t, t_{i+j+1} + 1] \subseteq P_{ij} \). It follows that \( \text{hold}(\sigma', P_{ij}) > \alpha C \) and the claim holds.

It follows that \( \text{hold}(\sigma', [\tau_i, \tau_{i+1}]) > \alpha C \). \( \{P_{ij}\} \) is a partition of \( [\tau_i, \tau_{i+1}] \) and, therefore,
\[
\text{cost}(\sigma, [\tau_i, \tau_{i+1}]) \geq \sum_{j=1}^{n} \text{hold}(\sigma', P_{ij}) > (p - 1)\alpha C.
\]

Lemma 4. Consider an ALG-round \( [\tau_i, \tau_{i+1}] \), such that either \( \sigma(\tau_i - 1) \neq \sigma'(\tau_i) \) or \( \sigma(\tau_i) \neq \sigma'(\tau_i) \). Then,
\[
\frac{\text{cost}(\sigma, [\tau_i, \tau_{i+1}])}{\text{cost}(\sigma', [\tau_i, \tau_{i+1}])} < k \left( 1 + \frac{1}{\alpha} \right) \quad \alpha \\leq 1
\]
\[
\frac{\text{cost}(\sigma, [\tau_i, \tau_{i+1}])}{\text{cost}(\sigma', [\tau_i, \tau_{i+1}])} < 1 + (k - 1)\alpha + k \quad \alpha \geq 1.
\]

Proof. ALG pays the setup cost \( C \) for a single transition during \( [\tau_i, \tau_{i+1}] \) (at \( \tau_i \)) and, therefore,
\[
\text{cost}(\sigma', [\tau_i, \tau_{i+1}]) = C + \text{hold}(\sigma', [\tau_i, \tau_{i+1}]).
\]

Substituting the ratio’s numerator from Lemma 2, we receive
\[
\frac{\text{cost}(\sigma, [\tau_i, \tau_{i+1}])}{\text{cost}(\sigma', [\tau_i, \tau_{i+1}])} \leq \frac{\text{hold}(\sigma', [\tau_i, \tau_{i+1}]) + pC((k - 1)\alpha + k)}{C + \text{hold}(\sigma', [\tau_i, \tau_{i+1}])}.
\]

Substituting the denominator from Lemma 3,
\[
\frac{\text{cost}(\sigma, [\tau_i, \tau_{i+1}])}{\text{cost}(\sigma', [\tau_i, \tau_{i+1}])} < 1 + \frac{pC(k + (k - 1)\alpha)}{C + (p - 1)\alpha C}.
\]
\[
= 1 + \frac{p(k - 1)\alpha + k}{1 + (p - 1)\alpha}.
\]

We denote \( \gamma(\alpha, p) \equiv 1 + \frac{p(k - 1)\alpha + k}{1 + (p - 1)\alpha} \). In order to compute \( p \) which produces the maximum ratio for a given \( \alpha \), we derive \( \frac{\partial \gamma}{\partial p} \). We get that \( \frac{\partial \gamma}{\partial p} = 0 \) for \( \alpha = 1 \), that is, the function is constant when \( \alpha = 1 \). For \( \alpha < 1 \), the function 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 \( \alpha < 1 \),
\[
\sup_{1 < \alpha < \infty} \gamma(\alpha, p) = \lim_{p \to \infty} \gamma(\alpha, p) = 1 + \frac{(k - 1)\alpha + k}{\alpha} = k \left( 1 + \frac{1}{\alpha} \right),
\]
whereas, \( \alpha > 1 \),
\[
\sup_{1 < \alpha < \infty} \gamma(\alpha, p) = \gamma(\alpha, 1) = 1 + (k - 1)\alpha + k.
\]

\[\Box\]

Theorem 2. The competitive ratio of DTrack-RR is bounded as follows:
\[
r(\text{DTrack-RR}) < k \left( 1 + \frac{1}{\alpha} \right) \quad \alpha \\leq 1,
\]
\[
r(\text{DTrack-RR}) < 1 + (k - 1)\alpha + k \quad \alpha \geq 1.
\]

Proof. We prove the upper bound on DTrack-RR’s competitive ratio for every OPT-round and conclude the same result for the entire run.

Consider the local ratio between the costs incurred by DTrack-RR and OPT during a single OPT-round \( [\tau_i, \tau_i] \), that is, \( \frac{\text{cost}(\sigma, [\tau_i, \tau_i])}{\text{cost}(\sigma', [\tau_i, \tau_i])} \). If either \( \sigma(\tau_i - 1) \neq \sigma(\tau_i) \) or \( \sigma(\tau_i) \neq \sigma'(\tau_i) \), the claim follows immediately from Lemma 4. Otherwise, \( \sigma(\tau_i - 1) = \sigma(\tau_i) = \sigma'(\tau_i) = \sigma' \). If DTrack-RR never transitions during the OPT-round, then...
\[
\text{cost}(\sigma, [r_i, r_{i+1}]) = \text{hold}(\sigma, [r_i, r_{i+1}]) = \text{hold}(\sigma', [r_i, r_{i+1}])
\]
\[
< \text{cost}(\sigma', [r_i, r_{i+1}]),
\]
and the claim trivially holds. Otherwise, let \(r_i < t < r_{i+1}\) be the first time after \(r_i\) such that \(\sigma(t) \neq \sigma^*\). Consider a schedule \(\sigma'\) that is obtained from \(\sigma^*\) by shifting the assignment to \(s^*\) from \(r_i\) to \(t\) (assume that this schedule is produced by some algorithm ALG). Note that \(\text{hold}(\sigma', [r_i, t]) = \text{hold}(\sigma, [r_i, t]) \geq 0\) and
\[
\text{cost}(\sigma, [t, r_{i+1}]) = \text{cost}(\sigma, [r_i, r_{i+1}]) - \text{cost}(\sigma', [r_i, t])
\]
\[
\geq \text{cost}(\sigma', [r_i, r_{i+1}]) - \text{cost}(\sigma', [r_i, t])
\]
\[
= \text{cost}(\sigma', [t, r_{i+1}]) \geq 0.
\]

By applying a well-known inequality \(\frac{p}{2} + \frac{q}{2} \leq \frac{p+q}{2}\) for \(0 \leq b \leq a\) and \(x \geq 0\) to the sought ratio, we get:
\[
\frac{\text{cost}(\sigma, [r_i, r_{i+1}])}{\text{cost}(\sigma', [r_i, r_{i+1}])} = \frac{\text{cost}(\sigma, [r_i, t]) + \text{cost}(\sigma, [t, r_{i+1}])}{\text{cost}(\sigma', [r_i, t]) + \text{cost}(\sigma', [t, r_{i+1}])}
\]
\[
< \frac{\text{cost}(\sigma, [r_i, t]) + \text{cost}(\sigma', [t, r_{i+1}])}{\text{cost}(\sigma', [r_i, t]) + \text{cost}(\sigma', [t, r_{i+1}])}.
\]
\[
\text{Since } s^* = \sigma(t-1) = \sigma'(t) \neq \sigma(t), \text{ the bound from Lemma 4 is applicable to the ALG-round } [t, r_{i+1}], \text{ and the claim follows.} \]

\[\square\]

### A.2 A Competitive Analysis of CTrack-RR

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

**Proof.** Consider an \(\text{OPT-round } [r_i, r_{i+1}]\) with \(p\) phases produced by CTrack-RR as defined in Appendix A.1, in which \(s^*\) is OPT's choice.

Consider a CTrack-RR round \([t, t']\) in which server \(s\) is CTrack-RR’s choice. If \(t < t' - 1\), then
\[
\text{cost}(\sigma, [t, t']) = \text{cost}(\sigma, [t, t' - 1]) + \text{cost}(s, t' - 1) 
\]
\[
\leq \text{cost}(\sigma, [t, t' - 1]) + aC.
\]
\[
\text{Since } t < t' - 1 \text{ and, hence, } \text{cost}(\sigma, [t, t' - 1]) \leq (\alpha + a)C, \text{ the result follows.}
\]

The rest of the proof is identical to that of Lemma 3. \(\square\)

### A.3 A Competitive Analysis of DTrack-B

**Theorem 4.** The competitive ratio of DTrack-B is bounded as follows:
\[
r(\text{DTrack}) < 1 + (k-1)\frac{\alpha}{\alpha-k} + k(\alpha-1)\frac{\alpha}{\alpha-k} \leq \beta \leq \beta \leq \alpha - 1, \quad (1)
\]
\[
r(\text{DTrack}) < k\left(1 + \frac{1}{\alpha}\right) \quad \alpha \leq 1 \text{ and } \beta \leq 0, \quad (2)
\]
\[
r(\text{DTrack}) < 1 + \frac{(k-1)\alpha}{\alpha-\beta} \max(0, \alpha-1) \leq \beta \leq \alpha. \quad (3)
\]

**Proof.** Consider the ratio between the costs incurred by DTrack-RR and OPT during a single OPT-round \([r_i, r_{i+1}]\). Similarly to the proof of Theorem 2, we derive
\[
\frac{\text{cost}(\sigma, [r_i, r_{i+1}])}{\text{cost}(\sigma', [r_i, r_{i+1}])} < 1 + \frac{p((k-1)\alpha + k)}{1 + (\alpha-1)\alpha} \max(0, \alpha-1) \min(\alpha, \alpha-\beta).
\]
\[
\text{If } \alpha \geq 1 \text{ and } \beta \leq \alpha - 1, \text{ then the derivative } \frac{d\varphi}{dp} \text{ is nonnegative and, hence,}
\]
\[
\sup_{1 \leq p < \infty} \varphi(\alpha, \beta, p) \geq 1 + \frac{p((k-1)\alpha + k)}{1 + (\alpha-1)\alpha} \min(\alpha, \alpha-\beta).
\]
\[
1 \leq p \leq \infty \text{ if } \alpha \geq 1 \text{ and } \beta \leq \alpha - 1. \quad (1)
\]
Theorem 5. The competitive ratio of DTrack-F and DTrack-B with $\alpha = \beta$ is $\Omega(C)$.

Proof. Assume wlog that $C$ is a positive integer (otherwise, the theorem can be proved for $C' = \lceil C \rceil$). Let $\varepsilon$ be a small number such that $0 < \varepsilon < \frac{\alpha}{1+\alpha}$. Consider three servers $s_0$, $s_1$, and $s_2$. Let $\text{hold}(s_2, t) = \varepsilon$ for all $t$, whereas $\text{hold}(s_0, t)$ and $\text{hold}(s_1, t)$ are defined as follows for integer values of $0 \leq i < \lfloor C/2 \rfloor$:

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

and

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

The hold costs during the interval $[0,3C+3]$ are depicted in Fig. 10. Note that, for $0 \leq i < \lfloor C/2 \rfloor$, it holds that $(2i+3)\varepsilon \leq (C+2)\varepsilon < \alpha$. Therefore, $\text{hold}(s_0, t) \leq \alpha$ and $\text{hold}(s_1, t) \leq \alpha$ for all $t$ during this interval. □

Lemma 6. Both DTrack-F and DTrack-B assign $s_0$ at times $t = 2i(C+1)$ and $s_1$ at times $t = (2i+1)(C+1)$ for $0 \leq i < \lfloor C/2 \rfloor$.

Proof. By induction on $i$. At time $t = 0$, both algorithms choose $s_0$ because it offers the minimal hold cost. The induction step considers two cases:

1. $t = (2i+1)(C+1)$. Both algorithms transitioned to $s_0$ at $2i(C+1)$ by the induction hypothesis. We compute $\text{def}(s_0, s_1, [2i(C+1), t])$ and

$$\text{def}(s_0, s_2, [2i(C+1), t]).$$

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

whereas

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

Note that both $\text{def}(s_0, s_1, [2i(C+1), t')$ and $\text{def}(s_0, s_2, [2i(C+1), t') are strictly smaller than $\alpha C$ for $t' < t$. Therefore, the transition happens at $(2i+1)(C+1)$ for the first time since $2i(C+1)$.  

2. $t = (2i+2)(C+1)$. This case is proved analogously to the previous one.

Upon each transition, DTrack-F selects the server with the zero hold cost, i.e., $s_0$ at times $2i(C+1)$ and $s_1$ at times $t = (2i+1)(C+1)$. DTrack-B selects the server that achieves the largest deficit, i.e., it makes the same choice. □
Consider a run of DTrack-F and DTrack-B during the interval \([0, C^2 - 1]\). Both algorithms behave identically. They transition \(C\) times during this interval (at \(t = i(C + 1)\) for \(0 \leq i \leq C - 1\)). Hence, the total setup cost is \(C^2\). The total hold cost exceeds \(\alpha C^2\) since a hold cost of above \(\alpha C\) is incurred between every two transitions. Hence, the total cost during the interval exceeds \((\alpha + 1)C^2\). In the same setting, OPT selects \(s_2\) at \(t = 0\) and never changes its assignment, thus paying a total setup cost of \(C\) and a total hold cost of \(\varepsilon(C^2 - 1) < \alpha C\). Hence, the competitive ratio of both online algorithms is \(\Omega(C)\).

\[\]