On the Remote Server Problem

or

More about TCP Acknowledgments

Leah Epstein[1⋆] and Alex Kesselman[2⋆⋆]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel.
Email: lea@math.haifa.ac.il
[2] Max Planck Institut für Informatik, Saarbrücken, Germany.
Email: akessel@mpi-sb.mpg.de

Abstract. We study an on-line problem that is motivated by service calls management in a remote support center. When a customer calls the remote support center of a software company, a Technician opens a service request and assigns it a severity rating. This request is then transferred to the appropriate Support Engineer (SE) who establishes a connection to the customer's site and uses remote diagnostic capabilities to resolve the problem. We assume that the SE can service at most one customer at time and a request service time is negligible. There is a constant set-up cost of creating a new connection to a customer's site and a specific cost per request for delaying its service that depends on the severity of the request. The problem is to decide which customers to serve first so as to minimize the incurred cost. This problem with just two customers is a natural generalization of the TCP acknowledgment problem. For the on-line version of the Remote Server Problem (RSP), we present algorithms for the general case and for a special case of two customers that achieve competitive ratios of exactly 4 and 3, respectively. We also show that no deterministic on-line algorithm can have competitive ratio better that 3. Then we study generalized versions of our model, these are the case of an asymmetric set-up cost function and the case of multiple SE's. For the off-line version of the RSP, we derive an optimal algorithm with a polynomial running time for a constant number of customers.

1 Introduction

Providing high quality support for company's products has a profound effect on customer satisfaction (see e.g. [11, 16]). A remote support center is intended to quickly identify and correct many problems with a software product without having to send a Technician to the site. Rapid escalation process ensures that the problem is quickly identified and solved to minimize equipment down time. When a call is accepted, the Technician troubleshoots the issue and classifies the severity of the request. The request is then assigned to one of the Support Engineers (SE's). A SE has to remotely access the customer's site in order to correct the problem. We assume that a SE can handle at

⋆ Research supported by Israel Science Foundation (grant no. 250/01).
⋆⋆ Research supported by AvH-Stiftung.

most one open connection simultaneously and the time required to service a request is negligible. Note that in real life it may take some time to resolve the problem. However, the main difficulty is rather to get access to the customer's machine, which is typically protected by FireWall or some other security software. Thus, a SE contacts the system administrator at the customer's site and opens a connection which is usually being monitored in order to obtain all necessary information about the environment and the product status. Then the problem can be quickly resolved (e.g. by fixing the configuration or installing a patch). We assume that there is a latency cost associated with delaying the service of a request that depends on its urgency and the type of the client's support contract and a set-up cost associated with creating a connection to the customer's site. The problem is to decide which customer to service in order to minimize the incurred cost. There is a trade-off between the latency and set-up costs. A set-up cost for a certain client is incurred only once when one or more requests of the client are serviced consecutively. Therefore, by accommodating a few customer's requests, the number of connections opened by a SE to the customer's site is reduced. However, delaying a request for a long time increases the latency cost, which can result in unacceptable level of service. Hence, there is a need to balance between the two costs.

We model the basic Remote Server Problem (RSP) as follows. There is a single server and k clients generating a sequence of requests σ. Time is continuous and at any moment a request for service from any client can arrive. We denote by σ_i a subsequence of requests generated by the i-th client and by σ_i^j the j-th request in this sequence. We also denote by n_i the total number of requests generated by the i-th client. We assume zero processing time for all requests. Thus, whenever the server opens a connection to a client, all pending requests of this client are serviced immediately. A server can maintain at most one open connection at time (if a new connection is created, the existing connection, if any, is closed). There is a constant set-up cost $r > 0$ of creating a new connection. The *schedule* S of a server is a sequence of established connections and their corresponding creation times. Let m be the total number of opened connections and let s_i^t be the latest time before or at time t at which the server is connected to client i, or otherwise let $s_i^t = 0$. For a request σ_i^j, we denote by a_i^j and d_i^j its arrival time and delay (waiting time), respectively. Note that σ_i^j is serviced at time $a_i^j + d_i^j = \min\{s_i^t : s_i^t \geq a_i^j\}$. Each request σ_i^j has an associated latency cost function f_i^j and the server incurs the latency cost of $f_i^j(d_i^j)$ on σ_i^j. We assume that a latency cost function f is a non-decreasing and continuous function of the delay value such that $f(0) = 0$. The goal of an algorithm is that of minimizing the incurred cost, that is

$$CF(S, \sigma) = \sum_{i=1}^{k} \sum_{j=1}^{n_i} f_i^j(d_i^j) + m \cdot r,$$

where $\sum_{i=1}^{k} \sum_{j=1}^{n_i} f_i^j(d_i^j)$ is the *latency cost* and $m \cdot r$ is the connection *set-up cost* of the schedule S. The problem can be extended to a situation where the set-up cost is a function of the client, i.e., each client i is associated with a set-up cost $r_i > 0$. In this case the term $m \cdot r$ is replaced by $\sum_{i=1}^{k} m_i r_i$ where m_i is the number of connections

opened to client i. Another extension is the Multiple-Server RSP, where there are s servers, each of which may have an open connection to one client.

We consider on-line algorithms for servicing requests, which learn about a new request only when it arrives to the system. Furthermore, we assume that the complete cost function is *unknown* to the online algorithm and the latency cost of a request is revealed with time (note that an existing minor problem can become blocking for the customer at any time). We use competitive analysis [18, 5] to evaluate the performance of our algorithms. In competitive analysis, the performance of the on-line algorithm is compared to the performance of on optimal off-line algorithm OPT, which knows in advance the entire sequence of all future requests. An advantage of competitive analysis is that a uniform performance guarantee is provided over all input instances. In the problem addressed in this paper, the algorithms seek to minimize their cost for a particular sequence of request arrivals. For an input sequence σ, denote the costs incurred by an online algorithm A and by OPT on σ by $CF(S^A, \sigma)$ and $CF(S^{OPT}, \sigma)$, respectively. We say that A is c-competitive if for every sequence of packets σ,

$$CF(S^A, \sigma) \leq c \cdot CF(S^{OPT}, \sigma) + a,$$

where a is a constant independent of σ.

Our results. For the on-line version of the basic RSP, we describe an algorithm called Balance that greedily serves a client when the cost of creating a connection equals to the latency cost incurred by its pending requests. We show that the competitive ratio of Balance is exactly 4 for the case of multiple clients. For the case of two clients, we derive an algorithm Two-Balance, which serves a client when the cost of creating a connection equals to *half* the latency cost incurred by its pending requests. We demonstrate that the competitive ratio of Two-Balance is 3. We also give a lower bound of 3 on the competitive ratio of any deterministic on-line algorithm, which holds even for the case of two clients. We note that this lower bound matches the upper bound for Two-Balance. Then we extend our analysis to the case in which clients may have different set-up costs and the case of multiple servers. For the case of different set-up costs, we propose an algorithm that achieves a competitive ratio of $6k - 2$ for k clients. We also modify Two-Balance to an algorithm Average-Two-Balance and show that it still has a competitive ratio of 3 in the case of two clients. We note that in this case, our lower bound of 3 holds for any given pair of set-up costs. For the case of k clients, we show a lower bound of $\sqrt{k-1}/2$ on the performance of any deterministic on-line algorithm. For the case of multiple servers, we present an algorithm that has a competitive ratio of at most $2(s + 1)$, where s is the number of servers, and demonstrate a lower bound of $3(s + 1)/2$ on the performance of any deterministic on-line algorithm. Finally, we give a polynomial-time algorithm for solving the off-line version of the RSP optimally for a constant number of clients.

The rest of the paper is organized as follows. We discuss the related work in Section 2. Algorithms for the on-line version of the RSP appear in Section 3. Section 4 contains the lower bounds. In Section 5 we study some natural extensions of our model. An algorithm for the off-line version of the RSP is presented in Section 6. We conclude with Section 7.

2 Related Work

Our problem is most closely related to the TCP acknowledgment problem, which can be viewed as a generalization of the ski-rental problem (also known as Rudolph's ski rental problem). In the TCP protocol, there exists a possibility of using a single acknowledgment packet to simultaneously acknowledge multiple outstanding packets, thereby reducing the overhead of the acknowledgments. Dooly et al. [7] introduced the dynamic TCP acknowledgment problem in which the goal is to minimize the number of acknowledgments sent plus the sum of the delays of all data packets (the delay of a packet is the latency between the packet's arrival time and the time at which the acknowledgment is sent). They gave a deterministic 2-competitive algorithm for this problem and showed that this is the best possible competitive ratio achievable by a deterministic on-line algorithm. Karlin et al. [12] developed a randomized on-line algorithm for the TCP acknowledgment problem with competitive ratio of $\frac{e}{e-1}$. Seiden [17] and independently Noga [15] demonstrated that this bound is tight. Albers and Bals [1] derived tight bounds for a variation of the problem in which the goal is to minimize the number of acknowledgments sent plus the maximal delay incurred for any of the packets.

The dynamic TCP acknowledgment problem can be reduced to the Remote Server Problem with two clients as follows. Let the set-up cost of a connection be $r = 1$. Each arriving TCP packet immediately generates a service request with a linear latency cost function $f(d) = d$. Initially, all the requests are generated by the first client. When the server establishes a connection to the first client, all the pending packets are acknowledged and new requests are being generated by the second client and so forth. It is easy to see that the total cost of the schedule equals to the value of the objective function of the original TCP acknowledgment problem. Our problem generalizes the TCP acknowledgment problem in that there are multiple clients and an individual latency cost function is associated with each request.

Generalizations of the ski-rental problem have been studied extensively. Fleischer [10] considered the Bahncard problem in which the price of a ticket is discounted by a constant factor β for a predefined time from the date of purchase. The author provided a deterministic algorithm with a competitive ratio of $2/(1+\beta)$ in the general case and a randomized algorithm with a competitive ratio of $\frac{e}{e-1+\beta}$ in the case that the Bahncard never expires. This was generalized in [12] to the case where due dates are present. Azar et al. [3] studied a capital investment problem where the goal is that of minimizing the total production and capital costs when future demand for the product being produced and investment opportunities are unknown. Some of the results of [3] were later improved by Bejerano et al. [4] and Damaschke [6].

In the k-server problem there is a metric space M in which there reside k identical mobile servers. When a request at a point in M is received, one of the servers must move to this point. Our goal is to minimize the total distance moved by all servers while servicing the request sequence. Manasse et al. [14] gave a 2-competitive algorithm for two servers and proved that no deterministic on-line algorithm for k servers can be better than k-competitive. Koutsoupias and Papadimitriou [13] showed that the work function algorithm for the k-server problem has competitive ratio of at most $2k-1$. Alborzi et al. [2] studied the k-client problem, where each of k clients has at most one active request on a point in M and a single server must serve all the requests. When

the request of a client is served, that client may choose to introduce another request. They demonstrated that several algorithms are $(2k-1)$-competitive and showed that no on-line algorithm can have competitive ratio better than $\lg k/2$ for the makespan and total completion time cost functions. In our problem, unlike the k-server and the k-client problems, the server does not move physically to the client but rather opens a remote connection.

Divakaran and Saks [8] considered scheduling systems with unit size caches where reordering of requests is available and requests may have different set-up and processing times. They presented a $O(1)$-competitive on-line algorithm for the maximum flow time problem. Feder et al. [9] studied classical caching problem in which up to k requests can be reordered. They solved the off-line version of the problem and gave tight bounds for the on-line setting, namely $k - O(1)$ for deterministic algorithms and $\Theta(\log k)$ for randomized algorithms.

3 On-Line Algorithms

In this section we consider the on-line version of the RSP. We first describe algorithm Balance for the case of multiple clients. Then we present algorithm Two-Balance for the case of two clients.

3.1 Multiple Clients

In this section we consider the case of multiple clients. We show that algorithm Balance achieves a competitive ratio of at most 4. Intuitively, Balance tries to find an equilibrium between the connection set-up cost and the latency cost of the pending requests.

On-line algorithm for the RSP – Balance:

A time t open a connection to client i if the latency cost incurred by its pending requests equals to the set-up cost r. Ties between clients are broken arbitrarily.

In what follows, we fix an input sequence σ. Consider the schedules S^B and S^O of Balance and OPT, respectively. Let m_i be the number of connections opened by Balance to client i. Denote by t_i^j and by t'^j_i the time at which Balance opens and closes the j-th connection to client i and let $t'^0_i = 0$. By the definition of Balance, a connection is closed only in order to serve another client. We assume that the very last connections of Balance and OPT to each client i are immediately closed when the last request $\sigma_i^{n_i}$ is served. This does not change the cost of the algorithm since the algorithm pays for creating connections and not for closing them. We also assume that after the last request arrives, Balance serves all the pending requests.

We divide the schedule of Balance w.r.t. the client i into intervals $I_i^j = (t'^{j-1}_i, t'^j_i]$ in which Balance is first disconnected and then connected to this client (see Figure 1). An interval starts just after time zero, or after a connection to client i was closed. Note that Balance does not create any connection at time zero.

We partition the total cost of the algorithm into the sum of costs of the intervals. We denote the cost incurred by an algorithm A (Balance or OPT) on client i during

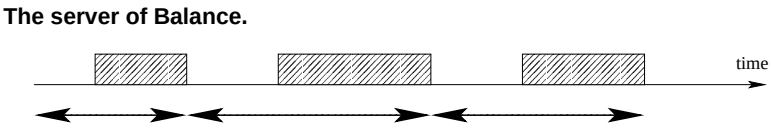

Fig. 1. An example of client's intervals sub-division.

interval I by $C_i(S^A, I)$. This cost includes the latency cost $CL_i(S^A, I)$ incurred by the pending requests of client i in S^A throughout I plus the connection set-up cost $CS_i(S^A, I)$, which is the cost incurred by the connections to client i that are *closed* during I (not the connections that are created during I). Note that Balance incurs the cost of $C_i(S^B, I_i^j) = 2r$ during any interval I_i^j since it incurs the latency cost of r and the connection set-up cost of r (while disconnecting from client i at the right endpoint of the interval). Hence, the total cost incurred by Balance is

$$CF(S^B, \sigma) = \sum_{i=1}^{k} \sum_{j=1}^{m_i} C_i(S^B, I_i^j) = \sum_{i=1}^{k} 2r \cdot m_i.$$

In a nutshell, we construct an assignment in which we assign the cost incurred by Balance to the intervals defined above so that the cost assigned to an interval is at most four times the cost incurred by OPT on the same client during this interval. The assignment routine is presented on Figure 2. Basically, we try to assign the cost $C_i(S^B, I_i^j)$ incurred by Balance during an interval I_i^j to the same interval. In case OPT did not incur sufficient cost during this interval, we will assign $C_i(S^B, I_i^j)$ to another interval on which the cost incurred by OPT is large enough and prove that it is always possible. Intuitively, we show that the situation that OPT did not incur sufficient cost during the interval can happen on at most half of the intervals.

For each interval I_i^j **Do** $ASG(I_i^j) = 0$;
For each interval I_i^j **Do**
  If $C_i(S^O, I_i^j) \geq r$ **Then** $ASG(I_i^j) = ASG(I_i^j) + C_i(S^B, I_i^j)$;
  Else $/* \; C_i(S^O, I_i^j) < r \; */$
    Let i' be the client to which Balance opens the j'-th connection at time t'^j_i after it closes the current connection to client i;
    $ASG(I_{i'}^{j'}) = ASG(I_{i'}^{j'}) + C_i(S^B, I_i^j)$;

(*) We denote by $ASG(I_i^j)$ the cost assigned to interval I_i^j.

Fig. 2. The assignment routine.

We say that an assignment routine is feasible if it is well defined, i.e., each interval is assigned an existing interval. We show that the assignment routine is feasible and assigns some cost only to intervals on which OPT has incurred a cost of at least r.

Lemma 1. *The assignment routine is feasible. For any interval I_i^j, if $ASG(I_i^j) > 0$ then $C_i(S^O, I_i^j) \geq r$.*

Proof. The lemma trivially holds if $C_i(S^O, I_i^j) \geq r$. Thus, suppose that $C_i(S^O, I_i^j) < r$. We will show that the assignment is well-defined, and $C_{i'}(S^O, I_{i'}^{j'}) \geq r$.

Client i' is clearly well defined unless we deal with the very last connection of Balance. Note that in this case $j = m_i$. We argue the cost incurred by OPT on the last interval of client i is at least r, and therefore $C_i(S^O, I_i^j) \geq r$. If the latency cost of OPT incurred during $I_i^{m_i}$ is at least r (i.e. $CL_i(S^O, I_i^{m_i}) \geq r$), we are done. Otherwise, OPT must be connected to client i at some point of time during $I_i^{m_i}$. Thus, by our assumption OPT disconnects from client i when the last request of this client arrives and incurs the connection set-up cost of at least r (i.e. $CS_i(S^O, I_i^{m_i}) \geq r$).

If $C_i(S^O, I_i^j) < r$, it means that OPT does not pay latency cost during I_i^j and does not disconnect from client i, since we have that $CS_i(S^O, I_i^j) < r$ and $CL_i(S^O, I_i^j) < r$. Therefore, OPT is either continuously connected to client i throughout I_i^j, or it connected to client i during I_i^j, at some time prior to t_i^j. In both cases, OPT is connected to i throughout $[t_i^j, t'^j_i = t_{i'}^{j'}]$. Assume that $CL_i(S^O, I_{i'}^{j'}) < r$, or otherwise we are done. This implies that OPT must be connected to client i' at some point of time during $(t'^{j'-1}_{i'}, t_{i'}^{j'}]$. Since OPT is connected to client i during $(t_i^j, t_{i'}^{j'}]$ and $t_i^j \geq t'^{j'-1}_{i'}$ by the construction of intervals, we get that it must have disconnected from client i' during $(t'^{j'-1}_{i'}, t_i^j] \subseteq (t'^{j'-1}_{i'}, t_{i'}^{j'}]$. Hence, we obtain that $CS_{i'}(S^O, I_{i'}^{j'}) \geq r$. ∎

The next theorem derives the competitive ratio of Balance by establishing an upper bound of four on the ratio between the cost assigned to an interval and the cost incurred by OPT during this interval.

Theorem 1. *The competitive ratio of algorithm Balance for the RSP is at most* 4.

Proof. Obviously, the total cost assigned by the assignment routine is $CF(S^B, \sigma)$. Lemma 1 implies that the assignment is feasible. Consider an interval I_i^j. We claim that $ASG(I_i^j) \leq 4 \cdot C_i(S^O, I_i^j)$. By Lemma 1, if $C_i(S^O, I_i^j) < r$ then I_i^j is not assigned any cost. In case $C_i(S^O, I_i^j) \geq r$, by the construction I_i^j can be assigned the costs incurred by Balance during I_i^j and another interval that is *uniquely* defined by I_i^j, that is interval $I_{i'}^{j'}$ of client i' from which Balance disconnects before it connects to client i. This constitutes at most the cost of $4r$ since Balance incurs the cost of exactly $2r$ on any interval.

It follows that

$$\begin{aligned} CF(S^B, \sigma) &= \sum_{i=1}^{k} \sum_{j=1}^{m_i} C_i(S^B, I_i^j) = \sum_{i=1}^{k} \sum_{j=1}^{m_i} ASG(I_i^j) \\ &\leq 4 \sum_{i=1}^{k} \sum_{j=1}^{m_i} C_i(S^O, I_i^j) \leq 4CF(S^O, \sigma). \end{aligned}$$

∎

3.2 Two Clients

In this section we consider the case of two clients. We describe algorithm Two-Balance and demonstrate that it is 3-competitive. Similarly to Balance, Two-Balance tries to balance the connection set-up cost and the latency cost, but it is not done symmetrically.

On-line algorithm for the two-client RSP – Two-Balance:

A time t open a connection to client i if the latency cost incurred by its pending requests equals twice the set-up cost (i.e. $2r$).

The analysis is similar to that of Balance. We use exactly the same assignment scheme and we again need to show that the assignment routine is feasible and assigns some cost only to intervals on which OPT incurred sufficiently large cost.

Lemma 2. *The assignment routine is feasible. For any interval I_i^j, that is not the interval corresponding to the very first connection of Two-Balance the following holds. (i) If $ASG(I_i^j) > 0$ then $C_i(S^O, I_i^j) \geq r$ and (ii) if $C_i(S^O, I_i^j) < r$ then $C_{i'}(S^O, I_{i'}^{j'}) \geq 2r$.*

Proof. The first part of the proof (including feasibility) is similar to that of Lemma 1. We will show that if for an interval I_i^j we have $C_i(S^O, I_i^j) < r$ then $C_{i'}(S^O, I_{i'}^{j'}) \geq 2r$. Note that OPT is connected to client i throughout $(t_i^j, t'^j_i]$ since $C_i(S^O, I_i^j) < r$. Given that there are only two clients, the connections of Two-Balance alternate between i and i', and we get that $t_i^j = t'^{j'-1}_{i'}$ and $t'^j_i = t_{i'}^{j'}$ (this is true because Two-Balance always disconnects from client i in order to connect to client i' and vice versa). It must be the case that $CL_{i'}(S^O, I_{i'}^{j'}) \geq 2r$ since $(t_i^j, t'^j_i] \subset I_{i'}^{j'}$ and the latency cost of the requests generated by client i' during $(t_i^j, t'^j_i]$ has reached exactly $2r$ by time t'^j_i. ▌

The next theorem derives the competitive ratio of Two-Balance. In our analysis we ignore the cost that might have been assigned to the interval of the second connection of Two-Balance, by the interval of the first connection of Two-Balance. This gives an additional additive constant of at most $3r$.

Theorem 2. *The competitive ratio of algorithm Two-Balance for the two-client RSP is at most* 3.

Proof. Again, the total cost assigned by the assignment routine is $CF(S^B, \sigma)$. Lemma 2 implies that the assignment is feasible. Consider an interval I_i^j, We claim that $ASG(I_i^j) \leq 3 \cdot C_i(S^O, I_i^j)$. We proceed by case analysis.

If $C_i(S^O, I_i^j) < r$, then by property (i) of Lemma 2, I_i^j is not assigned any cost at all.
If $r \leq C_i(S^O, I_i^j) < 2r$, then by property (ii) of Lemma 2, I_i^j can be assigned only the cost incurred by Two-Balance on I_i^j, which is at most $3r$.
If $C_i(S^O, I_i^j) \geq 2r$, I_i^j can be assigned the costs incurred by Two-Balance on I_i^j and another interval *uniquely* defined by I_i^j, which is at most $6r$.

It follows that $CF(S^B, \sigma) \leq 3CF(S^O, \sigma)$. ▌

4 Lower Bounds

In this section we deal with lower bounds. Remember that the complete cost function is *unknown* to the online algorithm and the latency cost of a request is revealed with time. First we present a lower bound of 3 on the performance of any deterministic on-line algorithm.

Theorem 3. *The competitive ratio of any deterministic on-line algorithm for the RSP is at least* 3.

Proof. We prove this theorem for a general setting, where the connection set-up costs of different clients may be distinct. Let A be a deterministic on-line algorithm. Suppose that there are two clients i and i', with set-up costs r and r'. Consider the following scenario. Initially, at time zero the first client generates a request. When A serves the request of the first client, the second client generates a request and so forth. This is repeated n times, so that the sequence contains n requests of each client. The latency cost function of a request increases linearly till it is served by A and remains constant thereafter, that is

$$f(d) = \begin{cases} d & d < x, \\ x & d \geq x, \end{cases}$$

where x is the delay of the request under A. The total cost incurred by A is

$$CF(S^A, \sigma) = \sum_{j=1}^{n} d_i^j + \sum_{j=1}^{n} d_{i'}^j + nr + nr'.$$

Without loss of generality, assume that $\sum_{j=1}^{n} d_{i'}^j \leq \sum_{j=1}^{n} d_i^j$. We define an off-line algorithm in the following way. The algorithm establishes a connection to the client i at time zero and keeps it open all the time. When the last request of client i is served, the algorithm creates a connection to the other client i' and serves all its pending requests. In addition, for each request $\sigma_{i'}^j$ of the client i' such that $d_{i'}^j > r + r'$, the algorithm opens a connection to client i' at time $a_{i'}^j$ upon arrival of $\sigma_{i'}^j$ and immediately re-opens back a connection to the client i after serving this request. Note that such a request incurs the connection set-up cost of $r + r'$. We get the following upper bound on the cost incurred OPT:

$$CF(S^O, \sigma) \leq \sum_{j=1}^{n} \min(d_{i'}^j, r + r') + r + r',$$

where the additive term of $r + r'$ is the connection set-up cost of the first and the last connections opened by the off-line algorithm and we can ignore it for large n. We show that

$$CF(S^A, \sigma) = \sum_{j=1}^{n} (d_i^j + r) + \sum_{j=1}^{n} (d_{i'}^j + r') \geq 2 \sum_{j=1}^{n} \left(d_{i'}^j + \frac{(r + r')}{2} \right)$$

$$\geq 3CF(S^O, \sigma) = 3 \sum_{j=1}^{n} \min(d_{i'}^j, r + r'),$$

which holds since for any $1 \leq j \leq n$ we have that

$$2(d_{i'}^j + \frac{(r+r')}{2}) \geq 3\min(d_{i'}^j, r+r').$$

The theorem follows. ∎

Next we show that the competitive ratio of Balance is at least 4.

Theorem 4. *The competitive ratio of algorithm Balance for the RSP is at least* 4.

Proof. Consider the following scenario. Assume that $k = 3$ and let M be a large even number and ϵ be a small constant. We define the latency cost function for all requests as

$$f(d) = \begin{cases} d & d < r, \\ r & d \geq r. \end{cases}$$

For $q = 0, 1, \ldots, M$, the first client generates a request at time $t = q \cdot (r + 2\epsilon)$. For the even values of q ($q = 0, 2, \ldots, M$), the second client generates a request at time $t = q \cdot (r + 2\epsilon) + \epsilon$ and for the odd values of q ($q = 1, 3, \ldots, M - 1$), the third client generates a request at time $t = q \cdot (r + 2\epsilon) + \epsilon$.

The schedule of Balance is presented on Figure 3. It first serves at time $t = r$ the request of the first client arriving at time $t = 0$. Then Balance serves at time $t = r+\epsilon$ the request of the second client arriving at time $t = \epsilon$. Thus, at time $t = r + \epsilon$ Balance will have an open connection to the second client. Thereafter, at time $t = 2r + 2\epsilon$ Balance serves the request of the first client arriving at time $t = r + 2\epsilon$ and at time $t = 2r + 3\epsilon$ it serves the request of the third client arriving at time $t = r + 3\epsilon$. The same situation repeats and so on.

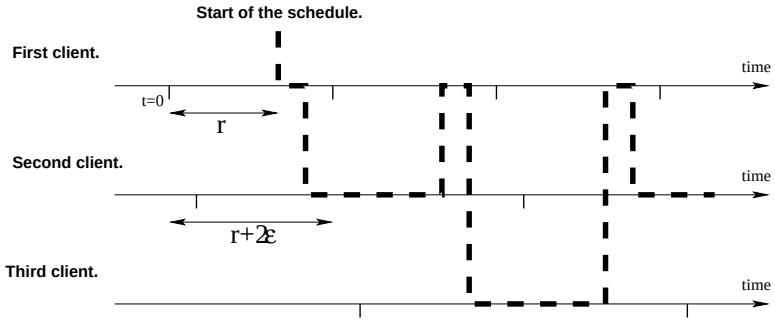

Fig. 3. The schedule of Balance.

The total cost incurred by Balance is at least $4Mr$ since on each interval $[q \cdot (r + 2\epsilon), (q + 1) \cdot (r + 2\epsilon)]$ for $q = 0, 1, \ldots, M$, it incurs twice the connection set-up cost of r plus twice the latency cost of r. On the other hand, OPT will keep a connection

to the first client open all the time during $[0, M(r+2\epsilon)]$ and serve the requests of the second and the third client at time $t = M(r+2\epsilon)$ incurring the total cost of at most $(M+3)r$. The obtained ratio can be made arbitrarily close to 4 for sufficiently large M. ∎

5 Extensions

In this section we consider some extensions of the basic RSP. We will study the case of asymmetric set-up cost function and the case of multiple servers.

5.1 Asymmetric Set-up Cost

In this section we consider the case in which client i has a connection set-up cost of r_i. We present algorithms that achieve competitive ratios of $6k-2$ and 3 for the case of k and two clients, respectively. We also show a lower bound of $\sqrt{k-1}/2$ on the performance of any deterministic on-line algorithm.

Definition 1. *We say that client i is* active *at time t if the server has an open connection to it at this time.*

Next we describe algorithm Max-Balance. The intuition behind Max-Balance is that we would like the server to stay connected to an active client with a high connection set-up cost till the latency cost incurred by a client with a low connection set-up is sufficiently high. This prevents the server from connecting too quickly to a "cheap" client and then paying a high connection set-up cost for re-connecting back to the "expensive" client.

On-line algorithm for the asymmetric RSP – Max-Balance:

A time t open a connection to client i if the latency cost incurred by its pending requests is larger or equal to the maximum of the set-up cost of the currently active client i' if any, and the set-up cost of client i (i.e. $\max(r_{i'}, r_i)$).

Theorem 5. *The competitive ratio of algorithm Max-Balance for the Asymmetric RSP is at most $6k-2$.*

Proof. We follow the lines of the analysis of Balance. The assignment routine appears on Figure 4.

Lemma 3. *The assignment routine for Max-Balance is feasible. For any interval I_i^j, $ASG(I_i^j) < (6k-2) \cdot C_i(S^O, I_i^j)$.*

Proof. Note that for each client $\hat{i}$, the last interval of $\hat{i}$ is an interval where OPT disconnects from it and therefore it incurs a connection set-up cost of at least $r_{\hat{i}}$. This shows that in assignments (3) and (4), there always exists a later interval as needed. In assignment (2), the existence of a next connection is proved as in Lemma 1.

Consider all possible assignments to one given interval I_i^j. By the construction, interval I_i^j can be assigned a value of at most $2C_i(S^O, I_i^j)$ each time when it is processed

For each interval I_i^j **Do** $ASG(I_i^j) = 0$;
For each interval I_i^j **Do**
  If $C_i(S^O, I_i^j) \geq C_i(S^B, I_i^j)/2$ **Then** $ASG(I_i^j) = ASG(I_i^j) + C_i(S^B, I_i^j)$ (1);
  Else / $*$ $C_i(S^O, I_i^j) < C_i(S^B, I_i^j)/2$ $*$ /
    If $C_i(S^B, I_i^j) = 2r_i$ **Then**
      Let i' be the client to which Max-Balance opens the j'-th connection at time t'^j_i after it closes the current connection to client i;
      If $C_{i'}(S^O, I_{i'}^{j'}) \geq r_i$ **Then** $ASG(I_{i'}^{j'}) = ASG(I_{i'}^{j'}) + C_i(S^B, I_i^j)$ (2);
      Else / $*$ $C_{i'}(S^O, I_{i'}^{j'}) < r_i$ $*$ /
        Let $I_i^{\hat{j}}$ be the first interval of client i s.t. $\hat{j} > j$ and $C_i(S^O, I_i^{\hat{j}}) \geq r_i$;
        $ASG(I_i^{\hat{j}}) = ASG(I_i^{\hat{j}}) + C_i(S^B, I_i^j)$ (3);
    Else / $*$ $C_i(S^B, I_i^j) > 2r_i$ $*$ /
      Let i' be the last client s.t. Max-Balance closes the j'-th connection to it at time $t'^{j'}_{i'} \leq t_i^j$ and $r_{i'} \geq CL_i(S^B, I_i^j)$;
      Let $I_{i'}^{\hat{j}}$ be the first interval of client i' s.t. $\hat{j} \geq j'$ and $C_{i'}(S^O, I_{i'}^{\hat{j}}) \geq r_{i'}$;
      $ASG(I_{i'}^{\hat{j}}) = ASG(I_{i'}^{\hat{j}}) + C_i(S^B, I_i^j)$ (4);

Fig. 4. The assignment routine for Max-Balance.

by any of assignments (1), (2), (3) and (4). Clearly, assignments (1) and (2) are feasible and the total value assigned by them to I_i^j is at most $4C_i(S^O, I_i^j)$ since they are associated with the intervals that are uniquely defined by I_i^j. We will demonstrate that assignments (3) and (4) are also feasible and the total value assigned by them to I_i^j is at most $6(k-1) \cdot C_i(S^O, I_i^j)$.

First consider assignment (3). We argue that it is feasible. Moreover, we will associate each assignment by (3) to a client i' and show that for any client $i' \neq i$, this assignment can be done at most once. Assume that the value of interval $I_i^{\hat{j}}$ of client i is assigned to interval I_i^j of the same client by assignment (3). Suppose that Max-Balance disconnects from client i and connects to client i' at time $t'^{\hat{j}}_i < t_i^j$. Note that OPT must be connected to client i at some point of time during $I_i^{\hat{j}}$ since $C_i(S^O, I_i^{\hat{j}}) < r_i$. Thus, I_i^j is the first interval during which OPT disconnects from client i after time $t'^{\hat{j}}_i$. Therefore, if Max-Balance during $(t'^{\hat{j}}_i, t'^{j-1}_i)$ connects to client i' after closing a connection to client i, it must be the case that the latency cost incurred by OPT on the corresponding interval of client i' is at least r_i. By the construction, in this case I_i^j is not assigned any value by assignment (3). We obtain that $ASG(I_i^j)$ can be increased at most $k-1$ times by assignment (3), one per each other client.

Now consider assignment (4). We claim that it is feasible and for any client $i' \neq i$, this assignment can be done at most twice. Let $I_{i'}^{j'}$ be the earliest interval of client i' whose value is assigned to interval I_i^j of client i by assignment (4). Note that if $C_{i'}(S^B, I_{i'}^{j'}) > 2r_{i'}$, then prior to time $t_{i'}^{j'}$, Max-Balance must be connected to a client whose connection set-up cost is at least $C_{i'}(S^B, I_{i'}^{j'})/2$ and suppose that it is the $\hat{j}$-th

connection of client i. If $\hat{j} = j$, then by the construction I_i^j will not be assigned the value of any subsequent interval of client i'. Otherwise, if $\hat{j} < j$, OPT is connected to client i at some point of time during $I_i^{\hat{j}}$ since $C_i(S^O, I_i^{\hat{j}}) < r_i$. Thus, I_i^j is the first interval during which OPT disconnects from client i after time $t'^{\hat{j}}_i$. Let $I_{i'}^{j''}$ be the latest interval of client i' whose value is assigned to I_i^j by assignment (4). We have that I_i^j will not be not assigned the value of any interval of client i' between $I_{i'}^{j'}$ and $I_{i'}^{j''}$ because on them OPT incurs the same latency cost as Max-Balance does. Moreover, I_i^j will not be assigned the value of any succeeding interval of client i'. We have that assignment (4) to I_i^j can be associated with at most two intervals of any specific client. Therefore, $ASG(I_i^j)$ can be increased at most $2(k-1)$ times by assignment (4). The lemma follows. ∎

The theorem follows directly from Lemma 3. ∎

We also sharpen the general upper bound for the case of two clients. The proof builds on that of Theorem 2. Let r_1 and r_2 be the set-up costs of the two clients, and let $r = (r_1 + r_2)/2$. We use algorithm Two-Balance with this value of r to create the schedule. We call this algorithm Average-Two-Balance.

Theorem 6. *The competitive ratio of algorithm Average-Two-Balance for the two-client Asymmetric RSP is at most* 3.

Proof. Consider an optimal off-line algorithm O_1 for the problem of two clients with set-up costs r_1 and r_2, and an optimal off-line algorithm O_2 for the problem where both clients have the same set-up cost r as defined above. We can assume without loss of generality that each one of those algorithms closes a connection only in order to open a connection to the other client, or when the algorithm terminates. Therefore, both of these algorithms have the property that the number of connections to the the first client differs by at most one from the number of connections to the second client. It is easy to see that a schedule of O_1 can be converted trivially to a schedule (not necessarily optimal) for the modified problem, possibly adding a constant term of $(\max\{r_1, r_2\} - \min\{r_1, r_2\})/2$ to the connection set-up cost. Observe that the latency cost remains the same. Since the cost of O_2 is not larger than the cost of the converted schedule, we get that

$$CF(S^{O_2}, \sigma) \leq CF(S^{O_1}, \sigma) + |r_1 - r_2|/2.$$

A similar argument can be applied to the difference between the cost of the on-line schedule using the set-up cost r and its real cost using the set-up costs r_1 and r_2. The theorem follows. ∎

Note that the above algorithm has the best possible performance for any pair of set-up costs, due to Theorem 3.

In the following theorem we show a lower bound of $\sqrt{k-1}/2$ on the performance of any deterministic on-line algorithm.

Theorem 7. *The competitive ratio of any deterministic on-line algorithm for the Asymmetric RSP is at least* $\sqrt{k-1}/2$.

Proof. Let A be a deterministic on-line algorithm and let $r = \sqrt{k-1}$. Suppose that r^2 clients have a connection set-up cost of 1 ("cheap" clients) and one client has a connection set-up cost of r ("expensive" client). Consider the following scenario. Time is divided into independent phases. At the beginning of a phase, each client generates a request. (The latency cost of all requests remains zero unless stated otherwise.) First, the latency cost of the request of the expensive client starts to grow linearly until A serves it. After the request is served by A, the latency costs remains at its current value. Then, if A has served all the requests of the cheap clients, we start a new phase. Otherwise, we arbitrarily choose one such request and its latency cost starts to grow linearly until it is served by A and then stops growing. At this time, a new request is generated by the expensive client and its latency cost grows till A serves it. The same situation is repeated until either A serves r requests of the expensive client or A serves all the requests of the cheap clients. Thereafter, we begin a new phase.

We obtain that A incurs the connection set-up cost of at least r^2 during a phase. There are two cases. If all "cheap" requests are served before the next phase starts, then clearly a connection set-up cost of 1 was paid by r^2 clients. Otherwise, the algorithm must connect to the "expensive" client r times, and so it pays the connection set-up cost of r at least r times, which again gives a total of at least r^2.

On the other hand, an off-line algorithm can just serve the r requests of the cheap clients that will be active in the future (i.e. the requests of the clients whose latency cost would not remain zero) at the beginning of a phase and then connect to the expensive client. incurring the connection set-up cost of $2r$ and zero latency cost. Thus, OPT incurs the total cost of at most $2r$. Note that all the unprocessed requests of the cheap clients will not incur any additional latency cost after the end of the phase. ∎

5.2 Multiple Servers

In this section we consider the case in which there are $s < k$ available servers. We propose an algorithm that has a competitive ratio of at most $2(s+1)$. Then we demonstrate a lower bound of $3(s+1)/2$ on the performance of any deterministic on-line algorithm.

Now we describe algorithm Round-Robin-Balance, which behaves exactly like Balance selecting servers in turn.

On-line algorithm for the Multi-Server RSP – Round-Robin-Balance (RR-Balance):

A time t open a connection to client i if the latency cost incurred by its pending requests equals to the set-up cost r using the next server in the Round-Robin order.

We demonstrate that RR-Balance is $2(s+1)$-competitive.

Theorem 8. *The competitive ratio of RR-Balance for the Multi-Server RSP is at most* $2(s+1)$.

Proof. We divide the schedule of RR-Balance into phases and the schedule of each client into intervals, similarly to Theorem 1. A phase is a collection of $s+1$ intervals of $s+1$ consecutive connections, which are associated with $s+1$ different clients. Note that RR-Balance incurs the cost of $2r$ on the corresponding interval of each client during such a phase.

We claim that OPT incurs the cost of at least r on one of those intervals. That is due to the fact that OPT either disconnects from one of the clients during one of the intervals of the phase and pays the connection set-up cost of r or pays the latency cost of r on the interval of the client to which it is not connected during its interval in the phase. ∎

Next we show a lower bound of $3(s+1)/2$ on the performance of any deterministic on-line algorithm.

Theorem 9. *The competitive ratio of any deterministic on-line algorithm for the Multi-Server RSP is at least* $3(s+1)/2$.

Proof. We consider a scenario similar to that of Theorem 3. Let A be a deterministic on-line algorithm. There are $k = s+1$ clients generating requests whose latency cost grows linearly until they are served. We select an inactive client, which generates a request and wait until this request is served by A (if all servers are in use, there is exactly one such client, otherwise one such client is chosen arbitrarily). This process continues until A serves in total n requests. Let D_i be the total cost (the connection set-up cost and the latency cost) for client i. Then $CF(S^A, \sigma) = \sum_{i=1}^{k} D_i$. Let i be a client for whom D_i is minimal. Clearly,

$$CF(S^A, \sigma) \geq kD_i = (s+1)\sum_{j=1}^{n_i}(d_i^j + r).$$

We define an off-line algorithm in the following way. The algorithm establishes connections to all clients but client i at time zero and keeps them open all the time. When the last request of some client is served and its server becomes free, the algorithm creates a connection to client i using this server and serves all its pending requests. In addition, for each request σ_i^j of the client i such that $d_i^j > 2r$, the algorithm opens a connection to client i at time a_i^j upon arrival of σ_i^j, disconnecting an arbitrary server for some client i', and immediately re-opens back a connection to the client i' after serving this request. Note that such a request incurs the connection set-up cost of $2r$. We get the following upper bound on the cost incurred by OPT:

$$CF(S^O, \sigma) \leq \sum_{j=1}^{n_i} \min(d_i^j, 2r) + kr,$$

where the additive term of kr is the connection set-up cost of the first and the last connections opened by the off-line algorithm ($k-1$ connections at time zero and one connection to client i to serve its pending requests). We can ignore this term for large n. We need to show that

$$CF(S^A, \sigma) \geq (s+1)\sum_{j=1}^{n_i}(d_i^j + r) \geq \frac{3(s+1)}{2} CF(S^O, \sigma) = \frac{3(s+1)}{2}\sum_{j=1}^{n} \min(d_i^j, 2r),$$

which holds since for any $1 \leq j \leq n$ we have that

$$2(d_i^j + r) \geq 3\min(d_i^j, 2r).$$

∎

6 An Optimal Off-Line Algorithm

In this section we present an algorithm that solves the off-line version of the RSP in a polynomial time for a constant number of clients. In what follows we fix an input sequence σ. The next lemma shows that any optimal schedule can be converted to a schedule with exactly the same cost in which a new connection is opened only when a new request in σ arrives and the first connection is created at time zero.

Lemma 4. *Given an optimal schedule S, it is always possible to construct an equivalent schedule S' s.t. $CF(S', \sigma) = CF(S, \sigma)$ in which a connection is opened only at the arrival time of some request in σ and the first connection, if any, is established at time zero.*

Proof. If the first connection in S is opened at time $t > 0$ then in S' we open it at time zero. Now we will iteratively modify S. Consider a connection to client i that is opened in S at time t s.t. no request in σ arrives at this time. Let i' be the client to which the server has been connected in S immediately before time t (note that i' is well-defined since it is not the first connection) and let $t' < t$ be the last time before t at which some request of either client i or the client i' arrives, or otherwise let $t' = 0$. We open a connection to client i in S' at time t' instead of time t. Since no request of client i' arrives during $[t', t]$, S' does not incur additional latency cost on the requests of client i'. Clearly, the latency cost incurred on the requests of client i can only decrease. By our construction, $CF(S', \sigma) \leq CF(S, \sigma)$. The lemma follows by the optimality of S. ∎

Now we present an algorithm that finds an optimal off-line schedule. According to Lemma 4, we can restrict our attention to schedules in which a connection is opened only when a new request arrives and the first connection is established at time zero. In a nutshell, we will construct a weighted directed graph G^σ in which a path between two designated nodes corresponds to a legal schedule and the length of the path is the cost of this schedule. Thus, the off-line RSP is reduced to the shortest path problem. For each request σ_i^j, we define a label $L_i^j = a_i^j$ and we also define a special label $L_0^0 = 0$. A node v in the graph is a product of k labels and the client id, that is $v = L_{c_1}^{j_1} \times \cdots \times L_{c_k}^{j_k} \times l$, where $l \in \{1, \ldots, k\}$. This node corresponds to the scheduler state at time $t_v = \max_i(a_{c_i}^{j_i})$, where $a_{c_i}^{j_i}$ is the time at which the last connection to client i was closed, if any, or $a_0^0 = 0$ otherwise and l is the id of the client to which the server is currently connected. There is a directed edge $e = (v, v')$ between nodes v and $v' = L_{c'_1}^{j'_1} \times \cdots \times L_{c'_k}^{j'_k} \times l'$ iff all of the following holds:

1. $t_{v'} \geq t_v$ (the time in a schedule is non-decreasing),

2. $l' \neq l$ (we never open a connection to the client that is already connected),
3. $L^{j_i}_{c_i} = L^{j'_i}_{c'_i}$ for $i \neq l$ (when we close a connection to the client l, the other clients remain unaffected),
4. $a^{j'_l}_{c'_l} = t_{v'}$ (we close a connection to the client l at time $t_{v'}$).

This edge corresponds to the situation at time $t_{v'}$ in which the scheduler closes a connection to client l and opens a connection to client l'. The weight of the edge is the connection set-up cost r plus the additional latency cost incurred by all the pending requests in σ during $[t_v, t_{v'}]$. Specifically, let $R_i = \{\sigma^j_i : a^{j_i}_{c_i} < a^j_i < t_{v'}\}$ for $i \neq l$ be the set of requests generated by client i before time $t_{v'}$ that have not been served yet by the scheduler. The latency cost of e equals to

$$\sum_{i \neq l, \sigma^j_i \in R_i} (f^j_i(t_{v'} - a^j_i) - f^j_i(\max(t_v - a^j_i, 0))).$$

Finally, we create a special source node v_s that has outgoing edges of weight r to all nodes $0 \times \cdots \times 0 \times l$ for $l \in \{1, \ldots, k\}$ (recall that the first connection is opened at time zero) and a special target node v_t that has incoming zero weight edges from each node v for which all the requests in σ are served, that is for each request σ^j_i either $a^{n_i}_i \leq a^{j_i}_{c_i}$ or $i = l$.

Optimal Off-Line Algorithm for the RSP:

Create the graph $G^\sigma = (V, E)$ as described above.
Find a shortest path P between v_s and v_t using the algorithm of Dijkstra for single source shortest path.
Transform P into a schedule S, that is for each edge $e = (v, v')$ in P s.t. $v' \neq v_t$, open a connection to client l' at time $t_{v'}$.

Consider the following example. Suppose that $k = 2$. The first client generates a request σ^1_1 with the latency function $f^1_1(d) = d$ at time $t = 1$ and the second client generates a request σ^1_2 with the same latency function at time $t = 2$. The corresponding graph is presented on Figure 5. Note that some nodes such as $0 \times 1 \times 2$ are unreachable from v_s because in an optimal schedule the server never opens a connection to the currently connected client. Two shortest paths between v_s and v_t are marked by the bold edges. In both schedules the server opens a connection to the first client at time zero ($v_s \to 0 \times 0 \times 1$) and at time $t = 1$ it serves the request of the first client. Then in the first and in the second schedules the server opens a connection to the second client at time $t = 1$ ($0 \times 0 \times 1 \to 1 \times 0 \times 2$) and at time $t = 2$ ($0 \times 0 \times 1 \to 2 \times 0 \times 2$), respectively, and serves the request of the second client at time $t = 2$. The cost of an optimal schedule is $2r$.

In the next theorem we show that the presented algorithm constructs an optimal schedule and has a polynomial running time for a constant number of clients.

Theorem 10. *The proposed algorithm finds a schedule S of minimum cost for the RSP in a polynomial time for a constant number of clients.*

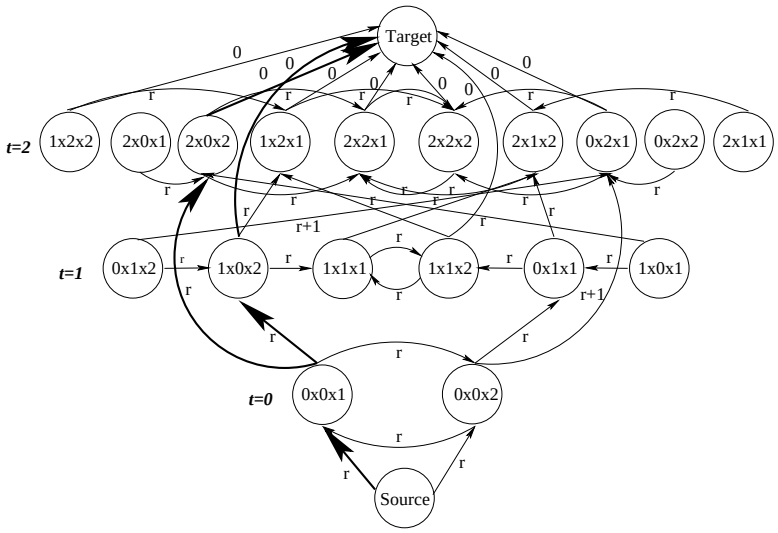

Fig. 5. An example of the graph G^σ.

Proof. First we demonstrate that the algorithm finds an optimal solution and then derive an upper bound on its running time.

By Lemma 4, there exists an optimal schedule S^O in which a connection is created only when a new request arrives and the first connection is opened at time zero. It is easy to see that there is one-to-one correspondence between such schedules and $v_s \rightarrow v_t$ paths in G^σ. Moreover, the cost of the schedule equals to the length of the corresponding path. Therefore, $CF(S, \sigma) = CF(S^O, \sigma)$.

Note that the number of nodes in G^σ is bounded by $|V| \leq (|\sigma| + 1)^k \cdot k + 2$, where $|\sigma|$ is the total number of requests in σ. Trivially, the number of edges in G^σ is at most $|E| < |V|^2/2$. The theorem follows since the running time of Dijkstra single source shortest path algorithm is $O(|V| \log |V| + |E|)$. ▌

We note that our algorithm can be easily extended to handle asymmetric set-up costs and multiple servers.

7 Conclusion and Open Problems

We have introduced a new on-line problem motivated by remote software support. A special case of the deterministic problem with just two customers is a generalization of the well-known TCP acknowledgment problem. We have presented upper and lower bounds for the basic version of the problem as well as for its natural extensions. Many of the established bounds are almost tight.

An interesting research direction can be to analyze a more realistic model in which requests may have non-zero service time. For the general variants of our model, the

lower bounds for deterministic algorithms turn out to be quite large. Thus, it would be interesting to consider randomized algorithms since worst-case analysis is too pessimistic. Another open problem is to close the gaps between the lower and the upper bounds for the basic model and for the case of asymmetric set-up cost function.

Randomized algorithms can be considered already for the basic problem of uniform set-up costs and a single server. For this problem a natural algorithm would be $Balance(\alpha)$ which picks a random parameter α according to some distribution, and then switches to client i once the total latency cost of the requests generated by client i equals α times the connection setup cost. The analysis of this algorithm is left for future research.

Acknowledgment. We would like to thank two anonymous referees for many helpful suggestions.

References

1. S. Albers and H. Bals, "Dynamic TCP Acknowledgement: Penalizing Long Delays," *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 47-55, 2003.
2. H. Alborzi, E. Torng, P. Uthaisombut and S. Wagner, *The k-Client Problem*, *J. Algorithms*, Vol. 41(2), pp. 115-173, 2001.
3. Y. Azar, Y. Bartal, E. Feuerstein, A. Fiat, S. Leonardi and A. Rosen, "On Capital Investment," *In Proc. 23rd International Colloquium on Automata, Languages, and Programming (ICALP96)*, Springer LNCS, Vol. 1099, pp. 514-525, 1996.
4. Y. Bejerano, I. Cidon and Joseph Naor, "Dynamic session management for static and mobile users: a competitive on-line algorithmic approach," *Proceedings of DIALM '00*, pp. 65-74, 2000.
5. A. Borodin and R. El-Yaniv, "On-Line Computation and Competitive Analysis," *Cambridge University Press*, 1998.
6. P. Damaschke, "Nearly optimal strategies for special cases of on-line capital investment," *Theoretical Computer Science*, Vol. 302(1-3), pp. 35-44, 2003.
7. D. R. Dooly, S. A. Goldman and S. D. Scott, "On-line analysis of the TCP acknowledgement delay problem," *JACM*, Vol. 48(2), pp. 243-273, 2001.
8. S. Divakaran and M. Saks, "An On-line Algorithm for the problem of single machine scheduling with job set-ups," *DIMACS Technical Report 2000-34*, 2000.
9. T. Feder, R. Motwani, R. Panigrahy, S. Seiden, R. van Stee and A. Zhu, "Combining request scheduling with web caching," *TCS, special issue in memoriam of Steve Seiden*, to appear.
10. R. Fleischer, "On the Bahncard problem," *Theoretical Computer Science*, Vol. 268(1), pp. 161-174, 2001.
11. IBM Remote Support, *http://www-1.ibm.com/services/us/index.wss/so/its/a1001373*.
12. A. R. Karlin, C. Kenyon and D. Randall, "Dynamic TCP acknowledgement and other stories about e/(e-1)," *ACM Symposium on Theory of Computing*, pp. 502-509, 2001.
13. E. Koutsoupias and C. H. Papadimitriou, "On the k-server conjecture," *Journal of the ACM*, Vol. 42(5), pp. 971-983, September 1995.
14. M. Manasse, L. A. McGeoch, and D. Sleator, "Competitive algorithms for server problems," *Journal of Algorithms*, Vol. 11, pp. 208-230, 1990.
15. J. Noga, *Private communication*.
16. Oracle Remote Support, *www.remote-dba.net*.

17. S. S. Seiden, “A guessing game and randomized online algorithms,” *In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pp. 592-601, May 2000.
18. D. Sleator and R. Tarjan, “Amortized Efficiency of List Update and Paging Rules,” *CACM 28*, pp. 202-208, 1985.