On Generalized Connection Caching

Susanne Albers*

Abstract

Cohen *et al*. [5] recently initiated the theoretical study of connection caching in the world-wide web. They extensively studied uniform connection caching, where the establishment cost is uniform for all connections [5, 6]. They showed that ordinary paging algorithms can be used to derive algorithms for uniform connection caching and analyzed various algorithms such as Belady's rule, LRU and *Marking* strategies. In particular, in [5] Cohen *et al*. showed that LRU yields a $(2k-1)$-competitive algorithm, where k is the size of the largest cache in the network. In [6], they investigated *Marking* algorithms with different types of communication among nodes and presented deterministic k-competitive algorithms.

In this paper we study *generalized connection caching*, also introduced in [5], where connections can incur varying establishment costs. This model is reasonable because the cost of establishing a connection depends, for instance, on the distance of the nodes to be connected and on the congestion in the network. Algorithms for ordinary weighted caching can be used to derive algorithms for generalized connection caching. We present tight or nearly tight analyses on the performance achieved by the currently known weighted caching algorithms when applied in generalized connection caching. In particular we give online algorithms that achieve an optimal competitive ratio of k. Our deterministic algorithm uses extra communication while maintaining open connections. We develop a generalized algorithm that trades communication for performance and achieves a competitive ratio of $(1+\epsilon)k$, for any $0 < \epsilon \leq 1$, using at most $\lceil 1/\epsilon \rceil - 1$ bits of communication on each open link. Additionally we consider two extensions of generalized connection caching where (1) connections have time-out values, or (2) the establishment cost of connections is asymmetric. We show that the performance ratio of our algorithms can be preserved in scenario (1). In the case of (2) we derive nearly tight upper and lower bounds on the best possible competitiveness.

1 Introduction

Cohen *et al*. [5] recently initiated the theoretical study of connection caching in the world-wide web. Communication between clients and servers in the web is performed using HTTP (Hyper Text Transfer Protocol), which in turn uses TCP (Transmission Control Protocol) to transmit data. The older HTTP/1.0 opened and closed a separate TCP connection for each transmission request. This caused congestion on the Internet because web pages typically contain inline images and other associated data which require a client to make multiple transmission requests to the same server within a short amount of time. The study by Cohen *et al*. is motivated by the fact that the new HTTP/1.1 works with *persistent connections*, i.e. connections are kept open so that they can be reused later. Persistent connections have a number of advantages. By opening and closing fewer TCP connections, CPU time is saved. Secondly, HTTP

*Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee 79, 79110 Freiburg, Germany. salbers@informatik.uni-freiburg.de

requests can be pipelined on a connection. Pipelining allows a client to make multiple transmission requests to the same server without waiting for each response. Thus a connection can be used more effectively. Finally, the network congestion is reduced by reducing the total number of packets that are used for TCP opens. Of course, each network node can maintain only a limited number of open TCP connections. A server or client keeps persistent connections open as long as its resources permit. If a connection is closed, there is a mechanism by which a client or server can signal the close [7]. In practice, servers often have a time-out value beyond which they will no longer maintain an unused connection [7].

Cohen *et al*. [5] introduced a theoretical model for connection caching that we will also adopt in this paper. The given network is modeled by an undirected graph G. The nodes of the graph represent the nodes in the network. The edges represent the possible connections. Each node has a cache in which it can maintain information on *open* connections. A connection $c = (u, v)$ is open if information on c is stored in the caches of both u and v. Otherwise the connection is *closed*. If a connection is open, we also say that it is *cached*. For a node v, let $k(v)$ denote the number of open connections that v can maintain simultaneously. The value $k(v)$ is also called the *cache size* of v. Let k be the size of the largest cache in the network. If v holds less than $k(v)$ open connections, then we say that it has an empty slot. For a connection $c = (u, v)$, let $cost(c)$ be the *establishment cost* of c that is incurred when c is opened. We assume that initially all connections are closed. An algorithm for connection caching is presented with a request sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_l$, where each request σ_t specifies a connection $c_t = (u_t, v_t)$, $1 \leq t \leq l$. A request σ_t can be served with cost 0 if c_t is cached. Otherwise, if c_t is not cached, there is a *miss* and c_t has to be opened at a cost of $cost(c_t)$. If there is no empty slot at node $w_t \in \{u_t, v_t\}$, then the algorithm has to close an open connection at w_t in order to make room for c_t. The goal is to serve the request sequence σ so that the total cost is as small as possible. We assume that at the end of σ, when all the requests are served, all open connections are closed again. In the model described here we assume that connections are *fully persistent*, i.e. they have no time-out values.

We are particularly interested in the development of *online* algorithms for the connection caching problem. An online algorithm has to serve each request without knowledge of any future requests. Given a request sequence σ, let $C_A(\sigma)$ denote the cost incurred by an online algorithm A in serving σ and let $C_{OPT}(\sigma)$ denote the cost paid by an optimal offline algorithm OPT. Following Sleator and Tarjan [12] A is called *c-competitive* if, for all request sequences σ, $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$, where a is a constant that must be independent of the request sequence.

Related work: Cohen *et al.* [5, 6] studied the connection caching problem assuming that the establishment cost is uniform for all connections. We refer to this problem as *uniform connection caching*. In [5] Cohen *et al*. first showed that the offline problem is APX-complete. They then observed that there is a close relationship between connection caching and *ordinary caching*, where one has to maintain a set of memory pages in a single cache so as to minimize the total access cost incurred on a sequence of requests to pages. In connection caching each node in the network can simply execute an algorithm for ordinary caching. Cohen *et al*. showed that any c-competitive algorithm for ordinary uniform caching, also known as paging, can be converted into a $2c$-competitive algorithm for uniform connection caching. Using Belady's optimal offline algorithm for paging [1], they obtained a 2-approximation for offline connection caching. Using the k-competitive online algorithm LRU (*Least Recently Used*), they obtained a $2k$-competitive online algorithm. Here it is assumed that if a node u closes a connection $c = (u, v)$, then v is not notified and only learns about the closing at the next request to c. Cohen *et al*. [5] also considered a stronger model where node v is notified and showed that LRU is $(2k - 1)$-competitive. Obviously, any

lower bound on the performance of ordinary caching algorithms also holds for connection caching algorithms. Thus no deterministic online algorithm for connection caching can be better than k-competitive and LRU was potentially a factor of 2 away from the optimum. In [6], Cohen *et al*. improved the upper bounds. More specifically, they considered *Marking* algorithms for connection caching and investigated different types of communication among nodes. They showed that any deterministic *Marking* algorithm is k-competitive and hence optimal if 1 extra bit is communicated per request. If no extra communication is allowed, then the algorithms achieve an upper bound of $2k - 1$.

Our contribution: In this paper we investigate *generalized connection caching*, where the establishment cost of connections can vary. For any connection c, $cost(c)$ can be an arbitrary positive value. Varying establishment costs occur in practice because the cost of establishing a connection depends for instance on the distance of the nodes to be connected and, more importantly, an the congestion in the network. Following [5] it is reasonable to assume that all TCP connections require the same socket buffer size in a network node. Thus we assume that all connections use the same space in cache and again denote by $k(v)$ the number of connections that node v can keep open simultaneously. Hence we work with connections of uniform size but varying costs. The corresponding ordinary caching problem is known as *weighted caching* and has been studied extensively, see e.g. [4, 10, 11, 13]. The offline variant of weighted caching can be solved in polynomial time [4]. The deterministic online algorithms *Balance* and *Landlord* are k-competitive [3, 4, 14]. The randomized algorithm *Harmonic* is k-competitive against any adaptive online adversary [11]. These competitive ratios are optimal. At this point, no randomized $o(k)$-competitive algorithm against any oblivious adversary is known for weighted caching.

In this paper we present a comprehensive study of generalized connection caching and, in particular, give optimal online algorithms. Clearly, a generalized connection caching algorithm can execute a weighted caching algorithm at each node. Extending the technique by Cohen *et al*. [5] in a straightforward way, one can convert any c-competitive algorithm for weighted caching into a $2c$-competitive algorithm for generalized connection caching. This immediately gives an upper bound of $2k$ on the best possible competitive ratio. In this paper we give tight or nearly tight analyses on the performance achieved by the currently known weighted caching algorithms when applied in generalized connection caching. We generally assume that if a node u closes a connection $c = (u, v)$, then v is notified, see [7]. We prove that the *Balance* algorithm, a popular algorithm for weighted caching, is not better than $(2k - 1)$-competitive. We then give an implementation of the *Landlord* algorithm, proposed for document caching in the web, and prove that it is k-competitive, and hence optimal, for generalized connection caching. (This result would also hold if connections had varying sizes.) *Landlord* uses *extra communication* when serving requests. By *extra communication* we refer to communication exchanged on an open connection in addition to that necessary for establishing and closing connections. We are able to reduce the amount of communication at the expense of increasing slightly the competitive ratio and formulate a trade-off. We develop a generalized algorithm *Landlord(ϵ)*, for any $0 < \epsilon \leq 1$, that is $(1 + \epsilon)k$-competitive and uses at most $\lceil 1/\epsilon \rceil - 1$ bits of extra communication for each open connection. We also analyze the randomized algorithm *Harmonic* and prove that it is k-competitive for generalized connection caching against any adaptive online adversary. Again, this competitive ratio optimal. Our implementation of *Harmonic* does not use extra communication; an additional feature of *Harmonic* is that it is memoryless. Hence we can achieve a competitiveness of k with either communication or randomization.

Additionally we consider two extensions of the connection caching model defined above. So far we

have assumed that connections are fully persistent and only have to be closed in order to make room for new connections. First, we address the extension that open connections have time-out values beyond which they will not be kept open. A time-out value of an open connection may be reset whenever the connection is requested again. We show that *Landlord*, *Landlord(ϵ)* and *Harmonic* can be modified so that their competitiveness is preserved. In the second extension we consider *asymmetric costs* [5], where the cost of establishing a connection $c = (u, v)$ can be different for u and v. We give nearly tight upper and lower bounds on the competitiveness that can be achieved in this scenario.

2 Lower bounds

Balance [4, 10, 14] is a very popular k-competitive online algorithm for ordinary weighted caching. Intuitively, *Balance* tries to distribute the loading cost of pages evenly among the k memory slots. When adapted to connection caching, the algorithm works as follows.

Algorithm Balance: For each node u in the network and for each of the $k(u)$ cache locations, the algorithm maintains a count which is initially 0. If there is a miss at a request to a connection $c = (u, v)$, then for each node $w \in \{u, v\}$ that does not have an empty slot, the algorithm evicts a connection that has the smallest count among the $k(w)$ slots. Ties may be broken arbitrarily. Connection c is opened and the count of each of the two cache slots holding c is increased by $cost(c)$.

We show that *Balance* is not better than $(2k - 1)$-competitive. This lower bound also holds if the establishment cost of a missing connection c is split among the two cache slots that will hold c, i.e. each count is increased by $cost(c)/2$.

Theorem 1 *Balance does not achieve a competitive ratio smaller than* $2k - 1$.

Proof: We construct a request sequence for which *Balance* does not perform well relative to an optimal offline algorithm OPT. The connections the request sequence is composed of are depicted in Figure 1.

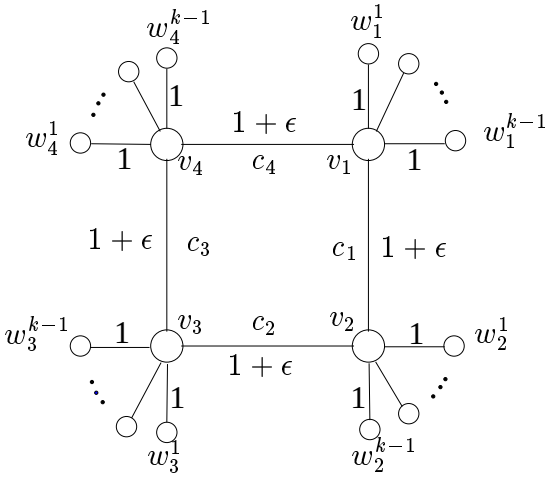

Figure 1: The requested connections

There are four *main nodes* v_i, $1 \leq i \leq 4$, for which there are four *main connections* $c_i = (v_i, v_{i+1})$, $1 \leq i \leq 3$, and $c_4 = (v_4, v_1)$. Each main node v_i has $k - 1$ neighbors w_i^j and associated connections

$d_i^j = (v_i, w_i^j)$, $1 \leq j \leq k-1$. Establishing a connection d_i^j incurs a cost of 1 while establishing a main connection incurs a cost of $1+\epsilon$, where $\epsilon > 0$ is an arbitrarily small value. Each main node has a cache of size k and each neighbor has a cache size of 1.

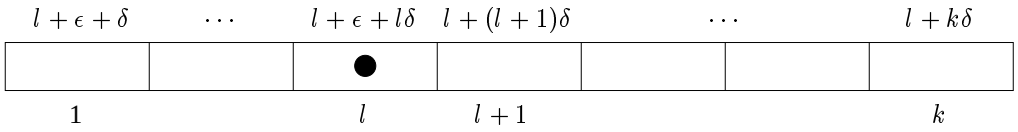

Figure 2: The configuration after phase l

When constructing the request sequence, we simultaneously keep track of the four caches at the main nodes. We assume that we start with the following cache configurations.

(1) Each v_i, $1 \leq i \leq 4$, holds the neighbor connections $d_i^1, \ldots, d_i^{k-1}$ in its first $k-1$ cache slots.

(2) The k-th cache slot at v_1 and v_2 holds c_1 while the k-th cache slot at v_3 and v_4 holds c_3.

(3) The count of the j-th cache slot at each main node is equal to $j\delta$ with $\delta = \epsilon/k$.

Such a configuration can easily be obtained as follows. At each v_i we first request $k-1$ *dummy connections*, where the j-th dummy connection has a cost of $j\delta$. We can assume without loss of generality that the j-th dummy connection is loaded into the j-th cache slot. We then present requests for c_1 and c_3, which are loaded into the k-th cache slot at each main node, and requests for all neighbor connections d_i^j, $1 \leq i \leq 4$ and $1 \leq j \leq k-1$. Neighbor connections $d_i^1, \ldots d_i^{k-1}$ are loaded into the first $k-1$ cache slots at v_i, $1 \leq i \leq 4$. At each main node, the j-th cache slot now has a count of $1+\delta j$. Reducing all counts by 1 does not affect the further execution of the algorithm.

Given a cache configuration with (1–3), we can construct a request sequence consisting of k phases such that the following holds.

(a) *Balance*'s cost in each phase is $4(k-1)+2(1+\epsilon) = 4k-2+2\epsilon$.

(b) The optimum offline cost in each phase is $2+2\epsilon$.

(c) After the k phases the cache configuration is identical to that described in (1–3).

Repeating the process infinitely many times, we obtain a lower bound on the competitive ratio achieved by *Balance* of $(2k-1+\epsilon)/(1+\epsilon)$, which can be arbitrarily close to $2k-1$.

We describe the construction of the k phases. Suppose that after l phases, $0 \leq l \leq k$, the following invariants hold.

(I1) Either c_1 and c_3 or c_2 and c_4 are cached in the l-th slot of the main nodes. Here slot 0 corresponds to slot k. Each main node has $k-1$ neighbor connections cached.

(I2) At each main node, the j-th cache slot has a count of $l+\epsilon+j\delta$ for $j = 1, \ldots, l$ and a count of $l+j\delta$ for $j = l+1, \ldots, k$.

The configuration of a main node's cache is shown in Figure 2. The bullet corresponds to a main connection. The $(l+1)$-st phase is as follows. If c_1 and c_3 are cached, then we issue requests to c_2 and c_4. Otherwise ,if c_2 and c_4 are cached, then we issue requests for c_1 and c_3. The connections are stored in the $(l+1)$st cache slot of each main node. After the two requests, at each v_i we present $k-1$ requests to neighbor connections. We always request the connection $d_i^* \in D_i = \{d_i^1, \ldots, d_i^{k-1}\}$ that is currently

not cached at v_i. The connections are first loaded into slots $l+2,\ldots,k$ and then into slots $1,\ldots,l$ provided that there is never an empty slot before the $(k-1)$-st request to a neighbor connection. This can be ensured by requesting the neighbor connections at the four main nodes almost simultaneously. If c_1 and c_3 were cached after phase l, then phase $l+1$ is $c_2c_4(d_1^*d_2^*d_3^*d_4^*)^{k-1}$. Otherwise phase $l+1$ is $c_1c_3(d_1^*d_2^*d_3^*d_4^*)^{k-1}$. Here $(S)^{k-1}$ denotes $k-1$ repetitions of request string S. In the j-th repetition, d_i^* is the neighbor connection currently not cached at v_i.

Note that after $k-1$ requests to neighbor connections at each main node, the main connection in slot l is evicted. This shows that (I1) holds after the phase. During the phase, at each main node the count of the $(l+1)$-st cache slot increases by $1+\epsilon$ while for the other cache slot it increases by 1. This proves (I2). Invariants (I1) and (I2) for $l=k$ and the fact a reduction of all counts by $k+\epsilon$ does not affect the further execution of the algorithm give a cache configuration identical to that described in (1–3). □

3 An optimal deterministic algorithm

A result by Manasse, McGeoch and Sleator [10] implies that no deterministic online algorithm for ordinary weighted caching can be better than k-competitive, where k is the number of pages that can be stored in cache. The same lower bound holds for generalized connection caching when $k=\max_u k(u)$.

We present an optimal deterministic k-competitive algorithm for generalized connection caching. The algorithm can be viewed as an implementation of a simplified *Landlord* algorithm [3, 14].

Algorithm Landlord (LL): For each cached connection c, the algorithm maintains a credit value $credit(c)$ that takes values between 0 and $cost(c)$. Whenever a connection is opened, $credit(c)$ is set to $cost(c)$. If there is a miss at a request to a connection (u,v), then for each node $w\in\{u,v\}$ that does not have an empty slot, execute the following steps. Let $\delta=\min_{c \text{ cached at } w} credit(c)$. Delete a connection c_w from w's cache with $credit(c_w)=\delta$ and decrease the credit of all the other connections cached at w by δ. Then open (u,v).

Note that for each cached connection $credit(c)\geq 0$ because if $credit(c)$ is decreased by δ, then $credit(c)\geq\delta$.

Landlord, as described above, is a centralized algorithm because the credit values of open connections are global shared variables. *Landlord* can also be formulated as a distributed algorithm. In a distributed implementation, for each open connection $c=(u,v)$, both endpoints u and v keep their own copies of $credit(c)$. If one endpoint, say u, reduced the credit by δ, then this change has to be communicated to v so that v can update its $credit(c)$ value accordingly.

In the following we will show that *Landlord* is k-competitive. In this paper, when proving upper bounds, we always use a slightly modified charging scheme for establishment costs: We charge the algorithms for connections that they *evict* rather than for connections that they *open*. A similar approach was taken in [11]. This charging scheme does not affect the performance of the algorithms because the initial and final caches are empty, i.e. all connections are closed. Hence the cost incurred for establishing connections equals the cost for closing connections. We will generally analyze online algorithms using a potential function Φ. The following lemma will be useful.

Lemma 1 *Let ON be a deterministic online algorithm that is analyzed using a non-negative potential function Φ which is initially 0. Suppose that, for all request sequences σ, the following statements hold.*

(1) For any request in σ, (a) when OPT serves the request, the increase in potential is bounded by c times the actual cost incurred by OPT and (b) when ON serves the request, the decrease in potential is at least as large as the actual cost incurred by ON.

(2) At the end of the request sequence, (a) when OPT closes a connection, the increase in potential is bounded by c times the actual cost incurred by OPT and (b) when ON closes a connection, the decrease in potential is at least as large as the actual cost incurred by ON.

Then ON is c-competitive.

Proof: The proof is standard in the study of amortized analysis. We simply sum up online and offline costs as well as potential changes over the whole request sequence and over the final operations when connections are closed. □

Lemma 1 is formulated for deterministic online algorithms. An analogous statement holds for randomized online algorithms against oblivious or adaptive adversaries. See [2] for a definition of the various types of adversaries. In this paper we will consider randomized online algorithms against adaptive online adversaries. In this case, when the adversary serves a request or finally closes a connection, the expected increase in potential must be bounded by c times the expected actual cost paid by the adversary. When the online algorithm serves the request, the expected decrease in potential must be at least as large as the expected actual cost incurred by the algorithm.

Theorem 2 *Landlord is k-competitive for generalized connection caching.*

Proof of Theorem 2: Let σ be an arbitrary request sequence. As mentioned above, we charge *Landlord* and an optimal offline algorithm OPT for connections that they evict and use a potential function Φ for the analysis of the algorithms. Define

$$\Phi = k \sum_{c \in S_{LL} \setminus S_{OPT}} credit(c) + \sum_{c \in S_{LL}} (cost(c) - credit(c)).$$

Obviously, Φ is non-negative. The initial potential is 0 because initially all connections are closed, i.e. S_{LL} and S_{OPT} are empty. We first show statement (1) of Lemma 1.

We consider an arbitrary request in σ. Let $c = (u, v)$ be the connection requested. If OPT does not evict a connection to serve the request, then the potential does not increase. If OPT does evict connections, then for every connection c' it evicts, the potential can increase by $k \cdot credit(c') \leq k \cdot cost(c')$ because c' might be in S_{LL}. However, OPT also pays a cost of $cost(c')$. Thus the increase in potential is bounded by k times the actual cost.

We next study *Landlord*'s service of the request. If c is cached by *Landlord*, then the actual cost is 0 and the potential does not change. So suppose that c is not cached. Opening the connection does not incur any cost and does not change the potential because $c \in S_{OPT}$ and $credit(c) = cost(c)$. We have to analyze the change in potential caused by evictions. Suppose that *Landlord* evicts connection $c_w = (w, x)$ at node $w \in \{u, v\}$. Connection c is cached by OPT but not by *Landlord* before *Landlord* serves the request. Since *Landlord* does not have an empty slot at w, there must exist a connection $c^*_w \in S_{LL} \setminus S_{OPT}$ cached by *Landlord* at w.

We first argue that the eviction of c_w causes a decrease of at least $k \cdot \delta$ in the first term of the potential function. If $c_w = c^*_w$, then the first term decreases by $k \cdot credit(c_w)$, which is $k \cdot \delta$ by the definition of δ.

If $c_w \neq c_w^*$, then $credit(c_w^*)$ decreases by δ causing a decrease of the first term of $k \cdot \delta$. For all the other connections cached at w that are not equal to c_w or c_w^*, a decrease of the $credit$-value can only result in a decrease of the first term.

In the second term of the potential function, the eviction of c_w causes a decrease of $cost(c_w) - credit(c_w) = cost(c_w) - \delta$ and an increase of $(k(w)-1)\delta \leq (k-1)\delta$ when the $credit$-values of the other $k(w)-1$ connections at w are decreased by δ. In summary, the total decrease in potential is at least $k \cdot \delta + cost(c_w) - \delta - (k-1)\delta = cost(c_w)$, which is the actual cost paid by *Landlord* on the request.

We still have to prove statement (2) of Lemma 1 and assume that first OPT and then *Landlord* closes all of its connections. For each connection c that OPT closes, Φ can increase by $k \cdot credit(c) \leq k \cdot cost(c)$, which is k times the actual cost paid by OPT. Whenever *Landlord* closes a connection c, the connection was already closed by OPT and hence $c \in S_{LL} \setminus S_{OPT}$. Thus the potential decreases by $k \cdot credit(c) + cost(c) - credit(c) \geq cost(c)$ and this is the actual cost paid by *Landlord*. □

4 Reducing the amount of extra communication

In the last section we presented the *Landlord* algorithm and also described a distributed implementation. The distributed implementation needs extra communication for maintaining open connections. If one endpoint of an open connection c reduces $credit(c)$ by δ using the *Landlord* strategy, then this update has to be communicated to the other endpoint. The amount of extra communication for an open connection can be large if the repeated δ reductions are small. We are able to reduce the amount of extra communication at the expense of increasing slightly the competitiveness of the algorithm. The main idea is that endpoints of an open connection c communicate updates in their credit values only when the total change since the last communication accumulates to $\epsilon \cdot cost(c)$, for some fixed $0 < \epsilon \leq 1$. We describe a fully distributed implementation of the algorithm.

Algorithm Landlord(ϵ): Let ϵ be a fixed constant with $0 < \epsilon \leq 1$. Each node u in the network executes the following protocol. For each open connection $c = (u, v)$, node u maintains its own credit value $credit(c, u)$ as well as an approximation of the corresponding credit $credit(c, v)$ maintained by v for that connection; this approximation is denoted by $\widetilde{credit}(c, v)$. Whenever a connection $c = (u, v)$ is opened, $credit(c, u)$ and $\widetilde{credit}(c, v)$ are set to $cost(c)$. Suppose that there is a miss at a request to a connection $c' = (u, w)$ and u does not have an empty slot. Let

$$\delta = \min_{c=(u,v) \text{ cached at } u} (credit(c, u) + \widetilde{credit}(c, v) - cost(c)).$$

Close a connection $c_u = (u, x)$ with $credit(c_u, u) + \widetilde{credit}(c_u, x) - cost(c_u) = \delta$ and, for any other connection c cached at u, reduce $credit(c, u)$ by δ. For each such connection $c = (u, v)$, node u also executes the following steps: If after the reduction by δ the credits satisfy $credit(c, u) + \widetilde{credit}(c, v) \leq cost(c)$, then c is closed. Otherwise u determines the smallest integers i and j such that $credit(c, u) > cost(c)(1 - i\epsilon)$ before the reduction and $credit(c, u) > cost(c)(1 - j\epsilon)$ after the reduction. It then sends $j - i$ update bits to v. Finally the requested connection c' is opened. At any time during the execution of the protocol, for each update bit that u receives from v on an open connection $c = (u, v)$, node u reduces $\widetilde{credit}(c, v)$ by $\epsilon \cdot cost(c)$ and closes c if $credit(c, u) + \widetilde{credit}(c, v) \leq cost(c)$ afterwards.

The next theorem expresses a trade-off between competitiveness and the amount of extra communication for each open connection.

Theorem 3 *For any $0 < \epsilon \leq 1$, Landlord(ϵ) is $(1+\epsilon)k$-competitive and uses at most $\lceil \frac{1}{\epsilon} \rceil - 1$ bits of extra communication for each open connection.*

Before we prove the theorem we mention some interesting points on the trade-off curve. Setting $\epsilon = 1$, we obtain a $2k$-competitive algorithm that does not use any extra communication. For $\epsilon = 1/2$, the resulting algorithm is $\frac{3}{2}k$-competitive and uses only one bit of extra communication. For $\epsilon \to 0$, the competitive ratio tends to k but the amount of communication can be arbitrarily large, as in the original *Landlord* strategy.

Proof of Theorem 3: We first show some properties of the credit values and their approximations.

Lemma 2 *Let $c = (u, v)$ be an open connection.*

a) The credits $credit(c, u)$ and $credit(c, v)$ as well as the approximations $\widetilde{credit}(c, u)$ and $\widetilde{credit}(c, v)$ take values between 0 and $cost(c)$.

b) For node u, $0 \leq \widetilde{credit}(c, u) - credit(c, u) < \epsilon \cdot cost(c)$ and a corresponding statement holds for node v.

Part b) of the above lemma implies that the approximations are always upper bounds on the true credits but at most an additive of $\epsilon \cdot cost(c)$ away from the true value.

Proof of Lemma 2: Part a): The credits of c satisfy $credit(c, u) + \widetilde{credit}(c, v) > cost(c)$ because the connection would be closed otherwise. This implies that the δ values in *Landlord(ϵ)* are always positive and hence $credit(c, u)$ is upper bounded by $cost(c)$. The same is true for $credit(c, v)$. Obviously $\widetilde{credit}(c, u)$ and $\widetilde{credit}(c, v)$ are upper bounded by $cost(c)$. If $credit(c, u)$ is reduced by δ, then $\delta \leq credit(c, u) + \widetilde{credit}(c, v) - cost(c)$ and hence $credit(c, u) - \delta \geq cost(c) - \widetilde{credit}(c, v) \geq 0$. Thus the credit values are lower bounded by 0. By part b), which we will show below, the approximations are always upper bounds on the true credits. Thus the approximations are also lower bounded by 0.

Part b): Consider one of the endnodes, say u, of the connection $c = (u, v)$. The proof is by induction. The inequality holds initially when the connection is opened because at that time $\widetilde{credit}(c, u)$ and $credit(c, u)$ are both set to $cost(c)$. Suppose that u reduces $credit(c, u)$ by δ. Let i and j be the smallest integers such that $credit(c, u) > cost(c)(1 - i\epsilon)$ before the reduction and $credit(c, u) > cost(c)(1 - j\epsilon)$ after the reduction. Before the reduction we have by induction hypothesis $\widetilde{credit}(c, u) \geq credit(c, u)$, which implies $\widetilde{credit}(c, u) \geq cost(c)(1 - (i - 1)\epsilon)$ because the approximations take values $cost(c)(1 - k \cdot \epsilon)$ for some integer k. Node v receives $j - i$ update bits and reduces $\widetilde{credit}(c, u)$ by $(j - i)\epsilon \cdot cost(c)$. Hence after the reduction $\widetilde{credit}(c, u) \geq cost(c)(1 - (j - 1)\epsilon)$, which is at least $credit(c, u)$ by the choice of j. This establishes the first inequality of part b). To prove the second inequality, we observe that before the reduction $credit(c, u) \leq cost(c)(1 - (i - 1)\epsilon)$ by the choice of i. The induction hypothesis implies $\widetilde{credit}(c, u) < cost(c)(1 - (i - 2)\epsilon)$ and hence $\widetilde{credit}(c, u) \leq cost(c)(1 - (i - 1)\epsilon)$ because the approximations only take values of the form $cost(c)(1 - k\epsilon)$ for some integer k. After u sent out $j - i$ update bits, $\widetilde{credit}(c, u) \leq cost(c)(1 - (j - 1)\epsilon)$ and the second inequality of part b) follows because $credit(c, u) > cost(c)(1 - j\epsilon)$ after the reduction. □

The bound on the extra communication follows from the next lemma.

Lemma 3 *Let $c = (u, v)$ be an open connection and assume that $\lceil \frac{1}{\epsilon} \rceil - 1$ update bits were exchanged so far. Then no endpoint of the connection will send out another update bit for c.*

Proof: Consider node u and suppose that it sent out $i \geq 0$ and received $\lceil \frac{1}{\epsilon} \rceil - 1 - i$ update bits. Then $credit(c,u) \leq cost(c)(1-i\epsilon)$ and $\widetilde{credit}(c,v) \leq cost(c)(1-(\lceil \frac{1}{\epsilon} \rceil - 1 - i)\epsilon)$. Node u can only send out another update bit if $credit(c,u)$ reaches or drops below the next threshold $cost(c)(1-j\epsilon)$ with $j > i$. Then

$$\begin{aligned} cost(c,u) + \widetilde{credit}(c,v) &\leq cost(c)(1-j\epsilon) + cost(c)(1-(\lceil 1/\epsilon \rceil - 1 - i)\epsilon) \\ &\leq cost(c)(2-((i+1)+\lceil 1/\epsilon \rceil - 1 - i)\epsilon) \\ &\leq cost(c) \end{aligned}$$

and u closes the connection instead. □

To evaluate the competitiveness of *Landlord(ϵ)*, we use the potential function

$$\begin{aligned} \Phi &= k \sum_{c=(u,v) \in S_{LL} \setminus S_{OPT}} (credit(c,u) + credit(c,v) + (\epsilon - 1)cost(c)) \\ &\quad + \sum_{c=(u,v) \in S_{LL}} (2cost(c) - credit(c,u) - credit(c,v)). \end{aligned}$$

Here S_{LL} denotes the set of open connections maintained by *Landlord(ϵ)*. The initial potential is 0 because all connections are closed in the beginning. During the execution of *Landlord(ϵ)*, for any open connection $c = (u,v)$, $credit(c,u) + \widetilde{credit}(c,v) > cost(c)$ and Lemma 2 part b) gives $\widetilde{credit}(c,v) - credit(c,v) < \epsilon \cdot cost(c)$. Adding these two inequalities we obtain

$$credit(c,u) + credit(c,v) + (\epsilon - 1)cost(c) > 0 \tag{1}$$

and thus the first sum of Φ is non-negative. The second sum is non-negative because the credit values of a connection are bounded by the cost, see Lemma 2 part a). In summary Φ is non-negative.

Consider an arbitrary request sequence. We prove statement (1) of Lemma 1. Let $c = (u,v)$ be the connection requested. For each connection $c' = (x,y)$ that OPT closes, the potential can increase by $k(credit(c',x) + credit(c',y) + (\epsilon - 1)cost(c')) \leq k(2cost(c') + (\epsilon - 1)cost(c') \leq (1+\epsilon)k \cdot cost(c')$, which is $(1+\epsilon)k$ times the actual cost paid by OPT. If OPT opens a connection, this cannot increase the potential because as shown in (1) the expression in the first sum of Φ is positive for each connection maintained by *Landlord(ϵ)*.

Suppose that *Landlord(ϵ)* evicts connection c_w at node $w \in \{u,v\}$ to serve the request. Since *Landlord(ϵ)* does not have an empty slot at w, there must exist a connection c_w^* cached at w with $c_w^* \in S_{LL} \setminus S_{OPT}$. We show that the eviction of $c_w = (w,z)$ and the subsequent reduction of credit values for connections at w cause a decrease of $k\delta$ in the first sum of Φ and an increase of at most $k\delta - cost(c_w)$ in the second sum of Φ. Thus the potential drops by at least $cost(c_w)$, which is the actual cost paid by *Landlord(ϵ)*. Here $\delta = credit(c_w,w) + \widetilde{credit}(c_w,z) - cost(c_w)$ is the value that determined the choice of c_w in *Landlord(ϵ)*. We first investigate the change in the first sum of Φ. If $c_w \neq c_w^*$, then the decrease of $credit(c_w^*, w)$ by δ results in a decrease of $k\delta$. If $c_w \neq c_w^*$, then the change in the first sum is

$$\begin{aligned} & -k(credit(c_w,w) + credit(c_w,z) + (\epsilon - 1)cost(c_w)) \\ \leq\ & -k(credit(c_w,w) + \widetilde{credit}(c_w,z) - cost(c_w)) \\ =\ & -k\delta. \end{aligned}$$

The first inequality follows because, by Lemma 2 part b), $credit(c_w, z) \geq \widetilde{credit}(c_w, z) - \epsilon \cdot cost(c_w)$. In the second sum of Φ, the reduction of $credit(c, w)$ values for connections $c \neq c_w$ cached at w, causes an increase of $(k_u - 1)\delta \leq (k-1)\delta$. The eviction of c_w results in a change of

$$\begin{aligned} & -2cost(c_w) + credit(c_w, w) + credit(c_w, z) \\ \leq \; & -2cost(c) + credit(c_w, w) + \widetilde{credit}(c_w, z) \\ = \; & \delta - cost(c_w). \end{aligned}$$

Thus the total increase is at most $k\delta - cost(c_w)$.

We analyze the other actions of *Landlord(ϵ)*. Sending update bits and changing approximations of credit values does not change the potential. If *Landlord(ϵ)* closes a connection $c = (u, v)$ because $credit(c, u) + \widetilde{credit}(c, v) \leq cost(c)$, then the second sum of Φ decreases by at least $cost(c)$ since $credit(c, v) \leq \widetilde{credit}(c, v)$. The first sum of Φ can only contribute another potential drop. Thus the total potential drop is at least as large as the actual cost incurred by *Landlord(ϵ)*.

It remains to prove statement (2) in Lemma 1. When OPT closes a connection, the potential can increase by at most $k(1+\epsilon)$ times the cost of the connection, which is $k(1+\epsilon)$ times the cost paid by OPT. When *Landlord(ϵ)* closes a connection $c = (u, v)$, the first sum of Φ drops by at least $credit(c, u) + credit(c, v) - cost(c)$ because the expression in the first sum is positive, $k \geq 1$ and $\epsilon > 0$. The second sum of Φ drops by $2cost(c) - credit(c, u) - credit(c, v)$. The total decrease in potential is at least $cost(c)$, which is the cost incurred by *Landlord(ϵ)*. □

So far we have reduced the amount of extra communiation at the expense of (slightly) increasing the competitive ratio. We can reduce the extra communication while preserving a competitiveness of k if we are willing to use randomization. We next give a randomized online algorithm that is k-competitive against any adaptive online adversary and does not use any extra communication bits. The competitiveness is optimal. No randomized online algorithm for generalized connection caching can be better than k-competitive against any adaptive online adversary. Our algorithm is an implementation of the *Harmonic* algorithm developed by Raghavan and Snir [11] for weighted caching. When applied in connection caching, *Harmonic* works as follows.

Algorithm Harmonic: If there is a miss at a request to a connection $c = (u, v)$, then for each node $w \in \{u, v\}$ that does not have an empty slot, the algorithm executes the following step. Let $D = \sum_{i=1}^{k(w)} 1/cost(c_i)$, where $c_1, \ldots, c_{k(w)}$ are the connections cached at w. Evict c_i with probability $p_i = 1/(D \cdot cost(c_i))$. Then open (u, v).

The algorithm has the additional interesting feature that it is memoryless.

Theorem 4 *Harmonic is k-competitive for connection caching against any adaptive online adversary.*

Proof: Let σ be an arbitrary request sequence. Again we charge *Harmonic* and the adaptive online adversary for the connections that they evict. Let

$$\Phi = k \sum_{c \in S_H \setminus S_{ADV}} cost(c),$$

where S_H and S_{ADV} are the sets of connections cached by *Harmonic* and the adaptive online adversary ADV.

Consider an arbitrary request to a connection (u, v) in the request sequence. Again we assume that ADV serves the request first and *Harmonic* serves the request second. When ADV serves the request, the potential can only increase if connections are evicted. For each connection c that ADV evicts, the potential can increase by at most $k \cdot cost(c)$, which is k times the actual cost incurred by the adversary.

When *Harmonic* serves the request, the potential can only change if connections are evicted because a connection opened by *Harmonic* must be in S_{ADV}. If *Harmonic* evicts a connection at node $w \in \{u, v\}$, then the expected cost incurred at w is $\sum_{i=1}^{k(w)} cost(c_i)/(D \cdot cost(c_i)) = k(w)/D \leq k/D$. Since *Harmonic* does not have an empty slot at w there must exist a connection $c_w^* \in S_H \setminus S_{ADV}$ cached at w. With probability $1/(D \cdot cost(c_w^*))$ this connection is evicted, causing a decrease in potential of $k \cdot cost(c_w^*)$. Thus the expected decrease in potential is at least k/D. This proves statement (1) of Lemma 1. Statement (2) is obvious. □

5 Extensions

5.1 Time-out of connections

In practice, servers usually have some time-out value beyond which they will not maintain an open unused connection [7]. Motivated by this fact, we study generalized connection caching in the setting that connections may expire. Kimbrel [9] studied ordinary paging and file caching when pages or files can expire.

We investigate the general setting where each cached connection has a time-out value. Whenever a connection c is opened at some time t, it is assigned a time t^{out}, $t^{out} > t$, at which the connection will expire. If the same connection is opened at times t_1 and t_2, where $t_2 > t_1$, then $t_2^{out} > t_1^{out}$. We assume that the time-out value of an open connection may be reset to a larger value whenever the connection is requested again. We show that the algorithms presented in this paper can be adapted easily to handle this situation.

Algorithm Time-Out: Use $A \in \{$*Landlord*, *Landlord(*ϵ*)*, *Harmonic*$\}$ as a basic caching strategy. If there is a miss at a request to a connection $c = (u, v)$, the algorithm *Time-Out* first evicts the expired connections at u and v. Then it executes strategy A.

Theorem 5 *The competitive ratio of Time-Out is equal to that of the basic strategy A.*

Proof: We present the analysis for strategy $A =$ *Landlord* and then sketch how to modify the analysis for *Landlord(*ϵ*)* and *Harmonic*.

In our analysis we may assume without loss of generality that OPT evicts connections immediately when they expire. An optimum offline strategy OPT that does not satisfy this property can be transformed into a strategy OPT' that follows this rule and does not incur a higher cost: Any expired connection c closed by OPT some time after expiration is closed by OPT' immediately when it expires; OPT' uses the empty cache slot only when OPT uses it. Clearly the incurred cost remains the same.

We generalize the analysis given in the proof of Theorem 2 and extend the potential function. Let S_{LL} and S_{OPT} denote the sets of connections cached by *Landlord* and OPT, respectively. This also includes connections that expired but have not been evicted from the caches. Let $S^{>}$ be the set of connections cached by *both Landlord* and OPT and for which OPT has a larger time-out value than *Landlord*. We

extend the potential function used in the proof of Theorem 2 by $\sum_{c \in S^>} credit(c)$, i.e.

$$\Phi \;=\; k \sum_{c \in S_{LL} \setminus S_{OPT}} credit(c) + \sum_{c \in S_{LL}} (cost(c) - credit(c)) + \sum_{c \in S^>} credit(c).$$

First note that a resetting of time-out values of open connections can only decrease the potential because the set $S^>$ can only become smaller.

We consider an arbitrary request in a request sequence and first analyze OPT's moves. The increase in potential that arises when OPT evicts a connection (be it expired or not expired) can be bounded in the same way as in the proof of Theorem 2 because the third term in Φ can only cause a decrease in potential. When OPT opens a connection c, two main cases have to be considered. (1) If $c \notin S_{LL}$, then the potential does not change. (2) If $c \in S_{LL}$, then the potential cannot increase: The first term of the potential function decreases by $k \cdot credit(c)$ and the third term may increase by $credit(c)$ if $c \in S^>$ afterwards.

We next analyze the influence of *Landlord*'s moves. Suppose that *Landlord* first evicts an expired connection c. If $c \in S_{OPT}$, then $c \in S^>$ since otherwise OPT would have evicted c when serving the request. Thus, the second and third terms of the potential function decrease by $cost(c) - credit(c) + credit(c) = cost(c)$, which is the actual cost incurred by *Landlord*. If $c \notin S_{OPT}$, then the potential decreases by $k \cdot credit(c) + cost(c) - credit(c) \geq cost(c)$. The eviction cost of a connection that is not expired can be bounded in the same way as in proof of Theorem 2 because the third sum in Φ can only cause a decrease in potential. When *Landlord* opens a connection c, then $c \in S_{OPT}$ and $c \notin S^>$ because OPT cannot have a larger time-out value than *Landlord*. Thus the potential cannot increase.

In the analysis of *Landlord(ϵ)* we extend the potential function by $\sum_{c=(u,v) \in S^>} (credit(c,u) + credit(c,v) + (\epsilon - 1)cost(c))$. In the analysis of *Harmonic*, we extend the function by $\sum_{c \in S^>} cost(c)$. □

5.2 Asymmetric costs

So far we assumed that the establishment cost of a connection (u, v) is independent of whether u or v has to establish the connection. In this section the study the case that the establishment cost does depend on the endnode. For any possible connection $c = (u, v)$, we have two consider two types of requests. By c_u we denote a request for connection c issued at u, i.e. u has to establish the connection if it not already open. Similarly, by c_v we denote a request for connection c issued at v, i.e. v has to establish the connection. Let $cost(c_u)$ and $cost(c_v)$ denote the establishment costs when u and v, respectively, have to open the connection. For any c, let $cost(c)_{\min} = \min\{cost(c_u), cost(c_v)\}$ and let $cost(c)_{\max} = \max\{cost(c_u), cost(c_v)\}$. Let $\mathcal{C}$ be the set of all possible connections in the graph. Define

$$\Delta = \max_{c \in \mathcal{C}} \frac{cost(c)_{\max}}{cost(c)_{\min}}.$$

Theorem 6 *No deterministic online algorithm for connection caching with asymmetric establishment cost can achieve a competitive ratio smaller then* $(k-1)\Delta + 1$.

Proof: We consider a very simple setting where we have a node u with cache capacity k and $k+1$ nodes $v_1, \ldots w_{k+1}$, each having a cache capacity of 1. An adversary constructs a request sequence consisting

of requests to connections (u, v_i), $1 \leq i \leq k+1$. The establishment cost of each of these connections is $\Delta \geq 1$ if u has to open it and 1 if v_i has to open it.

Let A be a deterministic online algorithm. In the request sequence constructed by the adversary, each request is made to the connection that is currently not cached by u. Thus, A has a miss on each request. The majority of the requests are issued at u; we call these requests *heavy*. However, a few requests will be issued at the corresponding node v_i; we call these requests *light*. Light and heavy requests are determined as follows. We partition the request sequence into *phases*. A phase ends when there have been requests to k distinct connections in the phase and a request to the $(k+1)$-st distinct connection occurs; the phase ends immediately before that request. In each phase, the first request is light. All the other requests are heavy.

The online algorithm has a cost of $(k-1)\Delta + 1$ in each phase. An optimal offline algorithm can serve the request sequence such that it has a miss only on the first request of the phase. Thus, its cost is 1 in each phase. □

The algorithms presented in Sections 3 and 4 can be applied to generalized connection caching with asymmetric establishment cost. In the description of the algorithms, $cost(c)$ refers to the cost that the algorithm has actually incurred when opening c.

Theorem 7 *The competitive ratios of Landlord, Landlord(ϵ) and Harmonic increase by a factor of Δ when used in generalized connection caching with asymmetric establishment cost.*

Proof: The analyses are almost the same as the original analyses in the proofs of Theorems 2, 3 and 4. In the definition of the potential functions, $cost(c)$ also refers to the cost that the online algorithm has actually incurred when opening c. The essential difference in the analyses is that if OPT or ADV evicts a connection $c = (u, v)$ for which it had paid an establishment cost of, say, $cost(c_u)$, then the potential might increase by $cost(c_v)$ because the online algorithm incurred another establishment cost. However, since $cost(c_u) \leq \Delta cost(c_v)$, the potential can increase by at most Δk times the actual cost incurred by OPT. □

6 Conclusions and open problems

In this paper we studied generalized connection caching and some extensions of the problem. We developed tight or nearly tight upper and lower bounds on the competitive ratio achieved by online algorithms. In particular we presented algorithms that use different amounts of extra communication. An interesting problem is to develop lower bounds in this context. What is the best possible competitiveness that can be achieved given a certain number of communication bits? We showed that an implementation of *Harmonic* is k-competitive against adaptive online adversaries. A challenging open problem is to develop ramdomized $o(k)$-competitive algorithms against oblivious adversaries. The problem is difficult because it involves designing $o(k)$-competitive algorithms for ordinary weighted caching. As for the offline variant of the problem, we can derive a polynomial time 2-approximation algorithm using the polynomial time offline algorithm for ordinary weighted caching, but no better approximation guarantees are known. Finally, an interesting topic is to investigate a combination of connection caching and *document caching* [8, 14], where one has to maintain local network caches with web documents, because the two problems arise together in practice.

Acknowledgment

We thank Edith Cohen, Haim Kaplan and Uri Zwick for interesting discussions and for sending us their paper [6].

References

[1] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78-101, 1966.

[2] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[3] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on International Technologies and Systems*, December 1997.

[4] M. Chrobak, H. Karloff, T. Paye and S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.

[5] E. Cohen, H. Kaplan and U. Zwick. Connection caching. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing*, pages 612-621, 1999.

[6] E. Cohen, H. Kaplan and U. Zwick. Connection caching under various models of communiation. In *Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 54–63, 2000.

[7] R. Fielding, J. Getty, J. Mogul, H. Frystyk and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. `http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html`

[8] S. Irani. Page replacement with multi-size pages and applications to Web caching. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 701–710, 1997.

[9] T. Kimbrel. Online paging and caching with expiration times. Manuscript, September 1998.

[10] M.S. Manasse, L.A. McGeoch and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208-230, 1990.

[11] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *Proc. 16th International Colloquium on Automata, Languages and Programming*, Springer Lecture Notes in Computer Science, Vol. 372, pages 687-703, 1989.

[12] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[13] N. Young. On-line caching as cache size varies. *Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 241–250, 1991.

[14] N.E. Young. Online file caching. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, 1998.