Connection Caching under Various Models of Communication

Edith Cohen
AT&T Labs-Research
180 Park Avenue
Florham Park, NJ 07932 USA
edith@research.att.com

Haim Kaplan
Computer Science Department
Tel-Aviv University
Tel-Aviv 69978, Israel
haimk@math.tau.ac.il

Uri Zwick
Computer Science Department
Tel-Aviv University
Tel-Aviv 69978, Israel
zwick@math.tau.ac.il

ABSTRACT

Motivated by Web applications, we recently introduced the following theoretical model for connection-caching: Each host on a network can maintain (cache) a limited number of connections to other hosts. A message can be transmitted from one host to another only if the connection between these two hosts is open, i.e., it is cached by both endpoints. If a message request arrives and the respective connection is not open (a *miss*), the connection needs to be established and certain activation cost is incurred. The establishment of the new connection may force the termination (eviction) of other connections at each endpoint.

The distributed nature of connection-caching makes it considerably more involved than the standanrd caching problem. It also makes it necessary to specify the type and amount of communication the different hosts are allowed to use to coordinate the contents of their caches. We consider three different models of communication. In the first and most basic model, hosts are allowed no extra communication. In particular, they are not notified when a connection is closed by its other endpoint. In the second model, hosts are notified of connection closures but are allowed no additional communication. In the third model, hosts are allowed to exchange information regarding shared open connections. The second model of communication corresponds to TCP under normal network conditions, the first corresponds to network failures. Although the third model is not yet implemented, we believe that it is interesting to explore its potential benefits.

We show that all algorithms belonging to a natural class of *marking algorithms*, working under the third communication model, are *optimally competitive*, i.e., achieve a competitive ratio of k, where k is the size of the largest cache in the network. This answers the main open problem left in [7]. Interestingly, all these optimally competitive algorithms exchange at most one extra *bit* per open connection. We also present an optimally competitive connection-caching algorithm that works under the second model. These results show that optimal competitiveness can be achieved with very limited communication.

Finally, we also consider *randomized* marking algorithms and show that they are $O(\log k)$-competitive.

1. INTRODUCTION

Clients and servers on the Web communicate by exchanging HTTP messages. HTTP messages are transported over TCP (Transmission Control Protocol) connections. When an HTTP request is issued, it can be transmitted on a previously-established connection, if such a connection exists, or it necessitates the establishment of a new TCP connection between the two endpoints. TCP is implemented on top of the connection-less IP (Internet Protocol), and the TCP connection information exists only at the two endpoints. (For more information about TCP/IP, HTTP, and their interaction see [14, 4, 9, 7].)

This reality imposes a tradeoff: On the one hand, as connection establishment induces latency and overhead, it is desirable to keep many connections open. On the other hand, each host can maintain only a limited number of open connections. An algorithm is therefore required to decide which connections to keep open and which ones to terminate.

Recently, we [7] proposed a theoretical model for *connection-caching* that generalizes the standard caching model. In this model each host has a cache of limited size which it uses to hold connections. A connection is considered open only if it is cached by *both* its endpoints. To serve a request for a connection (u, v) we have to open the connection (u, v), if it is not already open. Before opening the connection (u, v), each one of the endpoints u and v may have to shut down another connection, to make room in its cache for the new connection. (A precise definition of the connection caching model is given in the next section.) Connection-caching is an inherently distributed problem so it is necessary to specify the type and amount of communication the different hosts are allowed to use to coordinate the contents of their caches. We consider three different models of communication. In the first and most basic model, hosts are allowed no communication what so ever. In particular, they are not even notified when a connection is closed by its other endpoint. In this model, a connection (u, v) that was closed by v may still reside in the cache of u. In the second communication model, hosts are notified of connection closures but are allowed no additional communication. Here, when a connection is closed, it is immediately removed from the caches of both its endpoints. In the third model of communication, hosts are again notified of connection closures. In addition to that, hosts are now allowed to exchange (a minimal amount of) information regarding shared open connections. We refer to algorithms operating under these three models as *type 1*, *type 2* and *type 3* algorithms, respectively.

In [7], we observed that any standard cache replacement algorithm can be converted into a connection-caching algorithm by running the single-cache replacement algorithm locally at each node of the network. In fact, two different connection-caching algorithms are obtained, depending on whether hosts are notified of connection closures or not. If hosts are not notified, we obtain a type 1 algorithm and if they are notified, we obtain a type 2 algorithm. The casting of cache replacement algorithms as type 2 connection-caching algorithms is not always well-defined, as single-cache algorithms do not expect pages in their cache to disappear, as may happen in the connection-caching context, and therefore do not specify what to do in such a case. (This problem does not arise with type 1 algorithms.) Nevertheless, many natural paging algorithms give rise to natural type 2 connection caching algorithms when we require that (i) all holes in the caches are filled before initiating evictions (we refer to this requirement as *demand caching*), and (ii) when an eviction is necessary, the evicted connection is determined according to rule governing the behavior of the single-cache algorithm. When X is a standard caching algorithm, we use $\mathcal{X}_1$ and $\mathcal{X}_2$ to denote the type 1 and type 2 connection-caching algorithms derived from X.

In [7], we considered mainly type 1 and type 2 algorithms derived from single cache algorithms. We showed that any cache replacement algorithm (randomized or deterministic) with competitive ratio c has competitive ratio between c and $2c$ when cast as a type 1 connection-caching algorithm. The factor of 2 here is tight in the sense that there are single-cache algorithms that are c-competitive, while their corresponding connection-caching algorithm is only $2c$-competitive. This general result implies, for example, that $\mathcal{LRU}_1$, the type 1 algorithm derived from LRU (Least Recently Used) is $2k$-competitive. [1] We also proved in [7] that the competitive ratio of $\mathcal{LRU}_2$, the type 2 connection-caching algorithm derived from LRU, is exactly $2k-1$, where k is the size of the largest cache in the network. The question as to whether there exist connection-caching algorithms with better competitive ratios, and in particular, whether there exist connection-caching algorithms with a competitive ratio of k, as in the case of standard caching (Sleator and Tarjan [13]), was left open.

Type 2 communication models the normal termination of TCP connections when network round-trip-times are small with respect to connection open time, as connections should be terminated by an acknowledged message. Type 1 communication models operation under intermittent connectivity and crashes of host systems. Type 3 communication is not currently used for TCP connections, but we believe that its potential benefits are fundamental to the connection caching model. When a host chooses which connections to evict, it may benefit from evicting connections that are more likely to be evicted by the other endpoint. In reality, this information may be captured and communicated as load levels of different hosts. The questions that arise are whether communication helps, and if so, what type and by how much.

Our results: We define three families of deterministic *marking* connection-caching algorithms, one for each of the communication models. These policies are derived from marking algorithms for the standard caching problem (see [8]). We show that all marking connection-caching algorithms of type 1 and type 2 are $(2k-1)$-competitive, and more interestingly, that all type 3 marking algorithms are k-competitive. Type 3 marking algorithms exchange only a single bit of information per open connection. These bounds are tight, since $\mathcal{LRU}_1$ and $\mathcal{LRU}_2$ which constitute type 1 and type 2 marking algorithm, respectively, are known to be no better than $(2k-1)$-competitive. Furthermore, we also show that a type-2 marking algorithm called $\mathcal{FWF}$, which is derived from the single-cache algorithm FWF (Flush When Full) [10] is also k-competitive. FWF (and $\mathcal{FWF}$) evict pages as soon as they are unmarked, and hence, is not a demand-caching algorithm. These results answer the main open problem of [7].

Our results for deterministic connection-caching algorithms extend to the more general (k, h)-connection caching model, where at each node v with cache size $k(v)$ the offline adversary is only allowed to use a cache of size $h(v) \leq k(v)$.

Independently of our work, Albers [3, 2] recently considered connection caching where connections have varying establishment costs. Albers calls this problem *generalized connection caching*. Albers proves [3] that the Landlord algorithm proposed by Young [15] has competitive ratio of k. (When casted as a type 3 connection caching algorithm in our terminology.) The Landlord algorithm generalizes marking algorithms and hence this result generalizes our Theorem 3.2 to generalized connection caching. Albers also shows that the Harmonic algorithm [12] (which is randomized) is k competitive for generalized connection caching and that the competitive ratio of the BALANCE algorithm [6] is no better than $2k-1$.

We analyze randomized marking connection-caching algorithms in the three communication models. These algorithms are derived from the single-cache algorithm MARK introduced by Fiat et al [8]. Algorithm MARK maintains a single bit for each cached item, which is set when the item is requested. When an eviction is due, MARK simply selects a page to evict uniformly at random from all unmarked pages (if all pages are marked it first unmarks them all). Fiat et al [8] proved that MARK is $2H_k$-competitive in the single-cache setting. (A more complicated proof gives an upper bound of $2H_k - 1$ which is tight [1]). The general result of [7] implies that $\mathcal{MARK}_1$, is $4H_k$-competitive (in fact $4H_k - 2$ using the tight bound of [1]). We prove here that $\mathcal{MARK}_2$ is also $4H_k$-competitive. We note that the distributed na-

[1] It follows from the results of this paper that the competitive ratio of $\mathcal{LRU}_1$ is exactly $2k-1$.

	Deterministic Marking Algorithms	Randomized Marking Algorithms
Type 1	$2k-1$	$2(2H_k-1)$
Type 2	$2k-1$	$4H_k$
Type 3	k	k

Table 1: Upper bounds on the competitive ratios of all deterministic or randomized marking algorithms.

ture of connection caching made the analysis of $\mathcal{MARK}_2$ considerably more challenging than for the single-cache and type 1 setups. Our proof techniques incorporate several new ideas that are specific to the connection-caching model.

Another natural connection caching algorithm derived from MARK is a type 3 algorithm, denoted by $\mathcal{MARK}_3$, in which there is only one mark bit associated with a connection accessible by both endpoints. Curiously, we construct examples where $\mathcal{MARK}_3$ is $\Omega(k)$-competitive. The example demonstrates how the extra bit of communication allows the adversary to impose deterministic behaviour on $\mathcal{MARK}_3$. These results contrast with our results for deterministic marking algorithms, where the extra bit of communication provably helps.

The rest of this extended abstract is organized as follows. Section 2 gives some necessary definitions. In Section 3 we analyse the deterministic marking algorithms under the different communication models. In Section 4 we analyse the randomized marking algorithms under the different communication models. In Section 5 we analyse another general class of *conservative* connection-caching policies. Single-cache conservative algorithms include FIFO (First In First Out) and LRU and are known to have a competitive ratio of k [16, 5]. In Section 6 we highlight remaining open questions.

2. THE CONNECTION CACHING MODEL

We begin by reviewing the connection caching model. Let $G=(V,E)$ be an undirected graph representing a communication network. The vertices of V correspond to nodes in the network. The edges of E correspond to possible (virtual) connections between them. Each possible connection $e \in E$ may be either *open* or *closed*. At any given time, a vertex $v \in V$ may maintain at most $k(v)$ open connections. The parameter $k(v)$ is referred to as the size of the *connection cache* of v. The graph $H=(V,F)$ composed of all the open connections is therefore a subgraph of $G=(V,E)$ in which the degree of each vertex v is at most $k(v)$. We denote by k the size of the largest connection cache in the network, i.e. $k=\max_v\{k(v)\}$.

Initially all connections are closed. The network is then supposed to service a sequence $\sigma=\langle\sigma_1,\ldots,\sigma_\ell\rangle$ of *connection requests*. Each request $\sigma_i \in E$ is an edge in the graph $G=(V,E)$. If the connection σ_i is currently open, then the request can be serviced at no cost. This is called a *hit*. If the connection $\sigma_i=(u,v)$ is currently closed, then it must be opened (or reopened) to service the request. This is called a *miss*. Opening a connection incurs a fixed *activation cost*. If u already maintains $k(u)$ open connections, then one of these connections must be closed before the connection $\sigma_i=(u,v)$ is open. The same is true for v. Opening a connection may thus force the closure, and eviction, of two other connections. As described in the introduction we consider three versions of the connection-caching model that differ by the type of communication allowed between the hosts.

3. DETERMINISTIC MARKING

A single-cache marking algorithm acts as follows. Each page in the cache has a *mark bit* associated with it. When the bit is set we say that the page is *marked*. When this bit is cleared, we say that the connection is *unmarked*. When a page is inserted into the cache it is marked; only unmarked pages can be evicted; when all pages are marked and an eviction is due, all current pages get unmarked. When an unmarked page is requested and is currently in the cache, it *may* be marked by the algorithm.[2] LRU belongs to the class of marking algorithms where a page is always marked when requested. The policy FWF (Flush When Full) evicts pages as soon as they are unmarked. FWF is the weakest marking algorithm and constitutes the weakest member of that class (a page is in the cache of FWF only if it would have been cached under any other policy in that class). FIFO is a marking algorithm where a cached page is never marked when requested.

We consider three types of connection-caching marking algorithms, depending on the type of communication, and on whether local or global mark bits are used. In marking algorithms of type 1 and type 2 each vertex locally marks the connections in its cache. Thus, a connection (u,v) may be marked in cache(u), the cache of u, but unmarked in cache(v), the cache of v, or vice versa. With type 1 communication, vertices are not notified about connection closures. Thus, some of the connections held in cache(u), for example, may have already been closed but vertex u is still unaware of that. Type 2 algorithms make use of a stronger communication model. Connections are still marked locally, but vertices are notified of connection closures. If a connection (u,v) currently residing in cache(u) is *evicted*, i.e., closed, by v, then u is immediately notified, and the slot held by (u,v) in cache(u) becomes available for reuse. When a connection (u,v) is evicted by v, it is unmarked at v, but it may be marked or unmarked at u.

In marking algorithms of type 3, the mark bits are global. Each open connection (u,v) now has just a single mark bit, it is marked at cache(u) if and only if it is marked at cache(v). Furthermore, as vertices are notified of connection closures, the connection (u,v) resides in cache(u) if and only if it resides in cache(v). When a connection (u,v) is brought into the caches of u and v, it is marked. At some future stage, either u or v may be forced to unmark the connection (u,v). The other vertex is immediately notified that (u,v) was unmarked.

Of particular interest is the type 3 marking algorithm $\mathcal{FWF}_3$, which evict connection as soon as they are unmarked. $\mathcal{FWF}_3$ does not require additional communication for unmarkings,

[2]Our definition of marking algorithms is somewhat more general than previous definitions, since we do not require a requested unmarked cached page to be marked.

since they are communicated by connection closures. In fact, $\mathcal{FWF}_3$ is the same as its type 2 version, $\mathcal{FWF}_2$, which evicts a connection as soon as one of the endpoints unmarks it.

We shall need the following lemma that shows that the difference between the number of evictions of a particular algorithm to the number of misses it generates is bounded by a constant. We omit the proof which is easy by induction on the length of the sequence.

LEMMA 3.1. *Let miss(σ) be the number of misses, evict(σ) be the number of evictions, and hole(σ) be the number of holes remaining in all the caches of the network after servicing a sequence σ of connection requests using some connection caching algorithm. Also let $K = \frac{1}{2}\sum_{v \in V} k(v)$. Then*

$$miss(\sigma) \;=\; evict(\sigma) + (K - \frac{1}{2} \cdot hole(\sigma)) \,.$$

The main theorem of this section is the following.

THEOREM 3.2. *Any deterministic type 3 marking algorithm is k-competitive. (Recall that k is the size of the largest cache in the network.)*

PROOF. Let $\mathcal{ALG}$ be a type 3 marking algorithm. Let $\mathcal{OPT}$ be an optimal offline connection caching algorithm. We show that the number of connections unmarked by $\mathcal{ALG}$ is at most k times the number of misses of $\mathcal{OPT}$. The claim of the Theorem follows as each connection is unmarked before it is evicted, and as by Lemma 3.1, the number of misses of $\mathcal{ALG}$ is at most the number of evictions of $\mathcal{ALG}$, plus an additive constant.

Suppose vertex u is about to unmark all the connections in its cache as a result of a request for the connection (u, v). Consider the situation *after* $\mathcal{OPT}$ services this request but *before* $\mathcal{ALG}$ services it. At that moment, (u, v) resides in cache$_{\mathcal{OPT}}(u)$, the cache of $\mathcal{OPT}$ at u, but not in cache$_{\mathcal{ALG}}(u)$, the cache of $\mathcal{ALG}$ at u. As the caches of $\mathcal{OPT}$ and $\mathcal{ALG}$ at u are of the same size, there must be at least one connection (u, w) in cache$_{\mathcal{ALG}}(u)$ which is not in cache$_{\mathcal{OPT}}(u)$. We charge the unmarkings of the edges of u to the *most recent* request r that made $\mathcal{OPT}$ insert (u, w) into its cache. (As (u, w) is currently in cache$_{\mathcal{ALG}}(u)$, there must have been at least one request for this connection, and therefore (u, w) must have been inserted into cache$_{\mathcal{OPT}}(u)$ at some stage.)

Note that the request (u, v) that causes u to unmark its connections, may also cause v to unmark its connections. However, as the unmarkings at u are charged to a request for a connection (u, w), where $w \neq v$, and the unmarkings at v are charged to a request for a connection (v, w'), where $w' \neq u$, these unmarkings are charged to different misses of $\mathcal{OPT}$.

It remains to show that each miss of $\mathcal{OPT}$ is charged at most once with the unmarking of k connections by $\mathcal{ALG}$. We have already seen that a miss of $\mathcal{OPT}$ cannot be charged for unmarkings by two vertices that take place at the same time. We now show that a miss of $\mathcal{OPT}$ is also not charged for unmarkings by two vertices that take place at different times.

Let r be a request for connection (u, w) that causes a miss of $\mathcal{OPT}$. Let t_0 be the time in which r occurs. Let t_1, where $t_0 < t_1$, be the time of the first unmarking, by u or by w, for which r is charged. (If r is never charged then we are done.) After servicing the request at time t_1, the connection (u, w) is either out of cache$_{\mathcal{ALG}}(u)$ and cache$_{\mathcal{ALG}}(w)$, or is still present in these caches but is unmarked. The connection (u, w) is out of cache$_{\mathcal{OPT}}(u)$ and cache$_{\mathcal{OPT}}(w)$. Suppose, for contradiction, that r is also charged, by u or by w, at some later time t_2, where $t_1 < t_2$. At time t_2, the connection (u, w) must be in cache$_{\mathcal{ALG}}(u)$ and in cache$_{\mathcal{ALG}}(w)$ and it must be marked. As a connection is only marked as a result of a request, it follows that at least one request for (u, w) occurred between times t_1 and t_2. Let r' be the first such request. Note that $r' \neq r$. As (u, w) is out of cache$_{\mathcal{OPT}}(u)$ and cache$_{\mathcal{OPT}}(w)$ at time t_1, and as r' is the first request for (u, w) after time t_1, it follows that request r' is a miss for $\mathcal{OPT}$. Thus, at time t_2, the request r is not the most recent request that made $\mathcal{OPT}$ insert (u, w) into its cache. It follows that r will never be charged again. $\square$

THEOREM 3.3. *Any deterministic type 1 or type 2 marking algorithm is $(2k - 1)$-competitive.*

PROOF. The proof is a modification of the proof of Theorem 3.2. Let $\mathcal{ALG}$ be a type 1 or type 2 marking algorithm. Let $\mathcal{OPT}$ be an optimal offline connection caching algorithm. For a connection $e = (u, v)$, we let $\text{mark}_u(e)$ and $\text{mark}_v(e)$ be the mark bits of e in the caches of u and v, as assigned by ALG. If e is not in the cache of u, then we let $\text{mark}_u(e) = 0$. Similarly, if e is not in the cache of v, then $\text{mark}_v(e) = 0$. (Note that in type 1 algorithms, e may be in $\text{cache}_{\text{ALG}}(u)$ but not in $\text{cache}_{\text{ALG}}(v)$, or vice versa.) For the purpose of the proof, we also associate with each connection $e = (u, v)$ a *global mark bit* $\text{mark}(e) = \text{mark}_u(e) \wedge \text{mark}_v(e)$, i.e., the logical and of $\text{mark}_u(e)$ and $\text{mark}_v(e)$. An unmarking of e, by u or by v, that changes the global mark $\text{mark}(e)$ from 1 to 0, is said to be a *global unmarking* of e. Note that not every unmarking of e is a global unmarking of e, as the global mark of e may have been 0 before e is unmarked by, say, u. It is still true, however, that for any sequence of requests, the number of evictions carried out by $\mathcal{ALG}$ is at most the number of global unmarkings performed by $\mathcal{ALG}$. To prove the Theorem, we show that the number of global unmarkings performed by $\mathcal{ALG}$ is at most $2k - 1$ times the number of misses of $\mathcal{OPT}$.

Suppose vertex u is about to unmark all the connections in its cache as a result of a request for the connection (u, v). Consider the situation *after* $\mathcal{OPT}$ services this request but *before* $\mathcal{ALG}$ services it. At that moment, (u, v) resides in cache$_{\mathcal{OPT}}(u)$, the cache of $\mathcal{OPT}$ at u, but not in cache$_{\mathcal{ALG}}(u)$, the cache of $\mathcal{ALG}$ at u. As the caches of $\mathcal{OPT}$ and $\mathcal{ALG}$ at u are of the same size, there must be at least one connection (u, w) in cache$_{\mathcal{ALG}}(u)$ which is not in cache$_{\mathcal{OPT}}(u)$. Some of the markings going to be made by u are global unmarkings while some may be not. We charge the *global unmarkings* of the edges of u to the *most recent* request r

that made $\mathcal{OPT}$ insert (u, w) into its cache. (As (u, w) is currently in $\text{cache}_{\mathcal{ALG}}(u)$, there must have been at least one request for this connection, and therefore (u, w) must have been inserted into $\text{cache}_{\mathcal{OPT}}(u)$ at some stage.)

As in the proof of Theorem 3.2, it is easy to see that a miss of $\mathcal{OPT}$ cannot be charged for more than k global unmarkings that take place at the same time unit. We show further, that each miss of $\mathcal{OPT}$ can be charged for at most k global unmarkings that take place at some time t_1 and then for at most $k-1$ additional global unmarkings that take place at some time t_2, for some $t_1 < t_2$, and no more. The claim of the Theorem would then follow.

Let r be a request for connection (u, w) that causes a miss of $\mathcal{OPT}$. Let t_0 be the time in which r occurs. Let t_1, where $t_0 < t_1$, be the first time unit in which global unmarking are charged to r by either u or w. For concreteness, let us assume that these at most k global unmarkings were charged by u. It is easy to check, as in the proof of Theorem 3.2, that no additional global unmarkings performed by u will ever be charged to r. However, r may be charged for additional global unmarkings performed by w. Let t_2, where $t_1 < t_2$, be the first time unit in which global unmarkings initiated by w are charged to r. After servicing the request at time t_1, the connection (u, w) is unmarked at u. It is easy to see that no request for (u, w) can be made between t_1 and t_2. Thus, at time t_2, the connection (u, w) is either still unmarked at u, or is out of $\text{cache}_{\mathcal{ALG}}(u)$. Thus, the unmarking of (u, w) by w at time t_2 is *not* a global unmarking. It follows that at most $k-1$ global unmarkings are charged by w to r at time t_2. Again, it is easy to verify that r cannot be charged by additional global after time t_2. This completes the proof. $\square$

We can easily extend Theorems 3.2 and 3.3 to obtain bounds on the competitive ratios achieved by marking algorithms against offline connection caching algorithms that use smaller caches.

THEOREM 3.4. *Any deterministic type 3 marking algorithm is* $(\max_u \frac{k(u)}{k(u)-h(u)+1})$*-competitive against an offline connection caching algorithm which at every vertex u of the network is only allowed to use a cache of size $h(u) \le k(u)$.*

PROOF. The proof is a simple extension of the proof of Theorem 3.2. Suppose again that u is about to unmark the connections in its cache as a result for a request for a connection (u, v). Consider the situation *after* $\mathcal{OPT}$ services this request but *before* $\mathcal{ALG}$ services it. At that moment, (u, v) is in $\text{cache}_{\mathcal{OPT}}(u)$ but not in $\text{cache}_{\mathcal{ALG}}(u)$. As $\text{cache}_{\mathcal{ALG}}(u)$ is of size $k(u)$, while $\text{cache}_{\mathcal{OPT}}(u)$ is only of size $h(u)$, it follows that there are $d = k(u) - h(u) + 1$ connections $(u, w_1), (u, w_2), \ldots, (u, w_d)$ that are in $\text{cache}_{\mathcal{ALG}}(u)$ but not in $\text{cache}_{\mathcal{OPT}}(u)$. Let $r_1, r_2, \ldots, r_d$ be the most recent misses of $\mathcal{OPT}$ on the connections $(u, w_1), \ldots, (u, w_d)$. We charge each one of these misses of $\mathcal{OPT}$ by $k(u)/d = \frac{k(u)}{k(u)-h(u)+1}$ unmarkings of $\mathcal{ALG}$. As in the proof of Theorem 3.2, it is easy to verify that a miss of $\mathcal{OPT}$ is charged on at most one such occasion. $\square$

Using very similar arguments, we also get:

THEOREM 3.5. *Any deterministic type 1 or type 2 marking algorithm is*

$$(\max_{(u,v)\in E}[\frac{k(u)}{k(u)-h(u)+1} + \frac{k(v)-1}{k(v)-h(v)+1}]) - \textit{competitive}$$

against an offline connection caching algorithm which at every vertex u of the network is only allowed to use a cache of size $h(u) \le k(u)$.

Note that if $(u, v) \in E$ then also $(v, u) \in E$. Also note that even if $h(u) = k(u)$ for every $u \in V$, Theorem 3.5 is a slight strengthening of Theorem 3.3, as it states that any type 1 or type 2 marking algorithm is in fact $(\max_{(u,v)\in E}[k(u)+k(v)-1])$-competitive. In particular, a type 1 or type 2 marking algorithm may be $(2k-1)$-competitive, and not better, only on networks in which there are two adjacent vertices with caches of size k.

4. RANDOMIZED MARKING

As in Section 3 we define three randomized marking algorithms. The three algorithms differ by the type of connection caching used and whether we use a local or a global mark bits. Type 1 $\mathcal{MARK}$ has local mark bits and it works under the weak model of connection caching, type 2 $\mathcal{MARK}$ has local mark bits and works under the stronger connection caching model, and type 3 $\mathcal{MARK}$ has global mark bits and works under the stronger model of connection caching. All three algorithms evict an unmarked connection uniformly at random.

The following upper bound on the performance of type 1 $\mathcal{MARK}$ follows from theorem 4.4 in [7] and a tight bound on the performace of the marking algorithm for a single cache [1].

THEOREM 4.1. $\mathcal{MARK}_1$ *is* $(4H_k - 2)$*-competitive.*

4.1 Type 2 randomized marking

In this section we prove that $\mathcal{MARK}_2$ is $4H_k$-competitive. We note that the distributed nature of connection caching made the analysis of $\mathcal{MARK}_2$ considerably more challenging than for the single-cache and type 1 setups. Our proof techniques incorporate several new ideas that are specific to the connection-caching model.

THEOREM 4.2. $\mathcal{MARK}_2$ *is* $(4H_k)$*-competitive.*

PROOF. Let u be a vertex of the network. We break the sequence of requests that touch u into *phases* in the following way. The first phase begins with the first request. Each subsequent phase of u starts with a request that causes a miss when $\text{cache}(u)$ is full and all the connections in it are marked. Thus, the first request in each phase, except the first one, unmarks all the connections in $\text{cache}(u)$ and evicts one of them, chosen uniformly at random. Each phase but the last ends with a request that fills cache(u) with marked connections. Note, however, that not every such request finishes a phase, as some of the connections may be dumped from cache(u) before the next miss. The last phase ends with the last request in the sequence.

Consider a phase of the sequence of requests touching u. The connections that reside in cache(u) at the start of the phase are said to be *old* connections. All the other connections requested during the phase are *new* connections. We divide the requests that generate misses during the phase into three categories:

A First requests to new connections.

B Requests to connections that were already in cache(u) during the phase but are not currently in cache(u) becuase they were evicted, i.e., removed from cache(u) by u.

C Requests to connections that were already in cache(u) during the phase but are not currently in cache(u) becuase they were dumped, i.e., removed from cache(u) by their other endpoint.

We let A, B and C, respectively, be the sets of misses that belong to these three categories. We further break each one of these categories into two sub-categories, depending on whether the request caused an eviction of a connection from cache(u), or filled a hole in cache(u). We denote the resulting classes by A_1, A_2, B_1, B_2 and C_1, C_2, where the misses of A_1, B_1 and C_1 are the requests that cause evictions, while the misses of A_2, B_2 and C_2 fill holes.

We let evict(u) be the number of evictions from cache(u) during the phase. Clearly $\text{evict}(u) = |A_1| + |B_1| + |C_1|$. Our next goal is to show that $\text{evict}(u) \leq |A| + |B|$. To that end, we show that $|C_1| \leq |A_2| + |B_2|$. We prove, in fact, the slightly stronger result $\text{evict}(u) \leq |A| + |B'|$, where B' is the subset of the misses in B that occur while cache(u) still contains unmarked connections. To that end, we show that $|C_1| \leq |A_2| + |B'_2|$, where $B'_2 = B_2 \cap B'$.

We start by defining a one to one mapping $f : C \rightarrow A_2 \cup B_2 \cup C_2$. Let $r \in C$ be a request for a connection (u, v) that was dumped from cache(u) during the current phase. Let r' be the request that fills the hole caused by the dumping of (u, v). (We assume that r fills the hole created by dumping (u, v) if this hole still exists when serving r. Therefore r' must exists.) As r' fills a hole, we get that $r' \in A_2 \cup B_2 \cup C_2$. We map r into r', i.e., we let $f(r) = r'$. It is easy to see that f is indeed one to one. We note, that an element of C_2 may be mapped into itself.

We can now define a directed graph on $A_2 \cup B_2 \cup C$ in the following way. If r is mapped by f to r', we put in an edge from r to r'. As the indegree and outdegree of each vertex are at most one, this graph is composed of a collection of cycles and directed paths. Each request of C_1 starts a path in this graph. The endpoints of the paths in the graph must be from $A_2 \cup B_2$, as requests from C_2 are mapped by f to other requests. We can thus define a one to one mapping $g : C_1 \rightarrow A_2 \cup B_2$ in the following way: map each request $r \in C_1$ to the last request r' on the path that starts at r. Thus $|C_1| \leq |A_2| + |B_2|$, as required.

To obtain the stronger inequality $|C_1| \leq |A_2| + |B'_2|$, we note that if $r \in C_1$, then r causes an eviction from cache(u). It follows that when r arrives, there are still unmarked connections cache(u). It is not difficult to verify that if r is mapped by g into r', then r' occurs *before* r. Thus, there are still unmarked connection in cache(u) when r' arrives. It follows that g is a mapping from C_1 to $A_2 \cup B'_2$, and therefore $|C_1| \leq |A_2| + |B'_2|$, as required.

We should point out, at this point, that the partition of the requests touching u into phases, the sets A, B and C, and evict(u) are all random variables, as they are determined by the random choices made by u and by the other vertices in the network. Everything we said so far holds, however, for any random choice made by the algorithm.

We would now like to bound $E[\text{evict}(u)]$, the expected number of evictions at u during the phase. To that end, we want to bound $E[|A|]$ and $E[|B'|]$. Let N be the number of new connections accessed in the phase. (This is again a random variable as the length of the phase is random.) Clearly $|A| \leq N$, and thus $E[|A|] \leq E[N]$. We next turn our attention to bounding $E[|B'|]$.

The key observation used in bounding $|B'|$ is that if $r \in B'$, then r is the first request in the phase for an old connection (u, v) that was evicted from cache(u). The fact that (u, v) was evicted from cache(u) follows from the definition of B'. As (u, v) must have been unmarked when it was evicted, it follows that (u, v) is an old connection. The request r must be the first request to (u, v) during the phase, as after entering the cache (u, v) may be dumpted but it will never be evicted again until the end of the phase.

With each one of the k old connections of a phase we associate a *critical event*. The critical event corresponding to an old connection (u, v) is the first request for (u, v) during the phase, or the first dump of (u, v) during the phase, if it comes earlier. It is possible, of course, that neither of these happen, in which case no critical event is associated with (u, v). We consider an old connection as unmarked from the begining of the phase until the critical event associated with it happens or until the end of the phase. Note that the eviction of an old connection cannot be the critical event associated with a connection and we still consider the evicted old connection as marked.

It follows from the discussion above that $r \in B'$ if and only if r is a miss, there are still unmarked connections in cache(u), and r is the critical event corresponing to some old connection. We let X_j, where $1 \leq j \leq k$, be a random variable that gets the value 1 if the j-th critical event in the phase is a miss in B' (i.e. it is a miss occuring while there are still unmarked connections in cache(u).) Otherwise X_j takes the value 0. Clearly, $|B'| \leq \sum_{j=1}^{k} X_j$.

We now claim that the k-th critical event in a phase, if it happens, never causes a miss of type B', or in other words, that $X_k = 0$, always. If the k-th critical event is a dump, then we are done. Suppose, therefore, that the k-th critical event is the first request r in the phase for an old connection (u, v). If (u, v) is in cache(u) when r arrives, then we have a hit. Otherwise, since we know that all the other $k-1$ old connections were already requested during the phase, or were already dumped, there is no unmarked connection in cache(u). Thus, $r \notin B'$.

Let N_j be the number of new pages requested during the phase before the j-th critical event. Let U_j be the number of unmarked pages that are out of cache(u) when the j-th critical event of the phase occurs. (Recall that when an unmark connection is evicted from cache(u), we still consider it to be unmarked. However, if it is dumped we no longer consider it to be unmarked.)

It is straightforward to prove by induction on the request sequence of the phase that the number of unmarked and dumped old connections outside cache(u) is at most the number of new connection plus the number of holes. Since the number of dumped connections outside the cache is at least the number of holes it follows that for every j, $U_j \leq N_j \leq N$. Also, it is not difficult to see that the set of unmarked pages that are out of cache(u) is chosen uniformly at random from among all the subsets of size U_j of the unmarked connections. It follows that

$$E[X_j] \leq \frac{E[U_j]}{k-j+1} \leq \frac{E[N]}{k-j+1} .$$

By summing up, we get that

$$E[\sum_{j=1}^{k-1} X_j] \leq E[N] \cdot \sum_{j=1}^{k-1} \frac{1}{k-j+1} = E[N] \cdot (H_k - 1) ,$$

and therefore

$$E[\text{evict}(u)] \leq |A|+|B'| \leq E[N]+E[N]\cdot(H_k-1) = E[N]\cdot H_k .$$

By summing up over all the phases starting from the second (there are no evictions at the first phase if the caches are empty at the begining), we get that

$$E[\sum_i \text{evict}_i(u)] \leq E[N(u)] \cdot H_k ,$$

where $N(u) = \sum_{i>1} N_i$, and N_i is the number of new connections seen by u during phase i. It is easy to see that the total number of evictions plus the number of connections in the caches of $\mathcal{MARK}_2$ at the end of the sequence is an upper bound on the number of misses of $\mathcal{MARK}_2$.

The number of different connections that touch vertex u during phases $i-1$ and i, for every $i > 1$, is at least $k + N_i$. Therefore, the optimal offline policy must have had at least N_i misses incident with u during each pair of consecutive phases $i-1$, and i, for $i > 2$. Also if the caches are empty at the begining of the sequence then the number of misses of the optimal offline policy at the first two phases is at least $k + N_2$. Since each real miss may be counted in two different phases of two different vertices we conclude that the total number of misses of the optimal algorithm is at least $\frac{1}{4}(\cdot \sum_u N(u) + kn)$. The theorem now follows by deviding the upper bound on the number of misses of $\mathcal{MARK}_2$ by the lower bound on the number of misses of the optimal offline policy. □

4.2 Type 3 randomized marking

In this section we analyze the performance of the randomized marking algorithm with a global mark bit per connection. Clearly, Theorem 3.2 applies to this algorithm and hence type 3 randomized marking is k-competitive. Somewhat surprisingly, we show below a sequence on which the expected number of misses of type 3 randomized marking is k times bigger than the number of misses which the $\mathcal{OPT}$ incurs. This shows that for randomized marking the use of limited communication in the form of shared marked bits may cause the performance on the worst case to degrade compared to type 1 randomized marking that does not use any communication, and type 2 randomized marking that uses communication in a more limited form.

THEOREM 4.3. *The competitive ratio of $\mathcal{MARK}_3$ is at least k.*

PROOF. Let the set of nodes in our network be $V = \{x_i, y_i \mid 1 \leq i \leq k-1\} \cup \{v, w, z_1, z_2\},$. We now define a family of sequences consisting only of connections from the set $E = A \cup B \cup C \cup \{(v, z_1), (v, z_2), (w, z_1), (w, z_2)\}$ where $A = \{(x_i, y_j) | 1 \leq i, j \leq k-1\}$, $B = \{(v, x_i), | 1 \leq i \leq k-1\}$, and $C = \{(w, x_i), \mid 1 \leq i \leq k-1\}$. (See Figure 1.) We assume that all caches are of size k.

Each sequence starts with requests for all the connections in A, followed by two requests for connections (v, z_1) and (w, z_1). Then the l-th sequence in our family repeats the subsequence

$$s = B, (v, z_2), C, (w, z_2), B, (v, z_1), C, (w, z_1),$$

l time (by writing a set of connections in the sequence above we mean that each connection in the set is requested once in some fixed but arbitrary order).

It is easy to see that independently of its random bits type 3 randomized marking will miss every request in each of the sequences. It, in fact, acts deterministically since each time a node has to evict a connection it has only a single unmarked connection incident to it. Specifically, nodes $x_1, \ldots, x_{k-1}$ never unmark their incident connections and always have a single unmarked edge to evict. When x_i has to insert the connection (v, x_i) to its cache the connection (w, x_i) will be the only one which is unmarked (due to a previous unmarking at w) and a similar claim holds for v. On each request (v, z_1) or (v, z_2) vertex v unmarks its connections and similarly w unmarks its connections when it serves requests for (w, z_1) and (w, z_2).

The optimal offline algorithm can serve each of these sequences such that it incurs misses only on requests for the connections (v, z_1), (v, z_2), (w, z_1), and (w, z_2). Thus aside for initial misses the optimal offline algorithm incurs a single miss on every subsequence of k requests and the theorem follows. □

5. CONSERVATIVE ALGORITHMS

In this section we study other classes of algorithms generalizing LRU and FIFO other than the class of marking al-

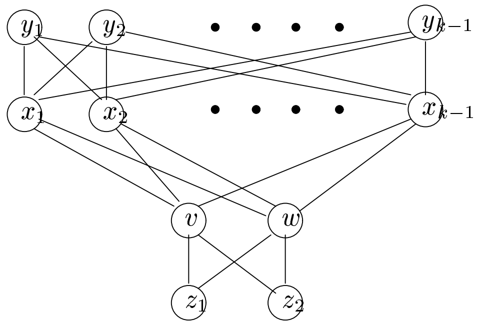

Figure 1: The connections requested by the family of sequences showing that the competitive ratio of type 3 randomized marking is at least k.

gorithms. We analyse the performance of these generalizations in the connetion caching model. We start by pointing out three different generalizations of FIFO for connection caching. Then we focus on Property 5.1 shared by all three generalizations of FIFO and type 1 and type 2 LRU. We then suggest a class of conservative algorithms analogous to the class suggested by Young [16] in the single-cache settings.

FIFO defines the age of an item stored in the cache as the time since the item last got into the cache. As LRU, in case of a miss FIFO evicts the oldest item in the cache. But in contrast with LRU, FIFO does not change the age of the cached item when it gets a hit. We can define three natural generalizations of FIFO for connection caching. The first is $\mathcal{FIFO}_1$, that is, each node independently uses FIFO under the weaker model of connection caching, we call this algorithm $\mathcal{FIFO}_1$. The second generalization, which we call $\mathcal{FIFO}'_1$ works as $\mathcal{FIFO}_1$ but in case of a miss, say on a request for connection (v, w), if (v, w) is in cache(w) then $\mathcal{FIFO}'_1$ changes the age of (v, w) in cache(w) to be zero, as if (v, w) was just inserted into cache(w). Similarly, $\mathcal{FIFO}'_1$ zeros the age of (v, w) in cache(v) if it is cached there. Notice that we update the age of a cached connection only in case of a miss, and therefore we change the age of at most one cached connection while serving a single request. The third generalization is $\mathcal{FIFO}_2$, namely FIFO deployed at the nodes in the second model of connection caching (where connections are evicted from both caches at once). Recall that an upper bound of $2k-1$ on the performance of $\mathcal{FIFO}_2$ follows from Theorem 3.3. All three algorithms $\mathcal{FIFO}_2$, $\mathcal{FIFO}_1$, and $\mathcal{FIFO}'_1$ as well as type 1 and type 2 $\mathcal{LRU}$ satisfy the following property.

Property 5.1. *Between the time a connection (u, v) enters cache(u) and the time it is evicted by u, there must be requests to at least k other connections touching u. (More specifically, there must be requests for connections $(u, v_1), \ldots,$ (u, v_k), where the v_i are all distinct and different from v.)*

Before proving an upper bound on the competitive ratio of any algorithm satisfying Property 5.1 we introduce the following definition. Let (v, w) be a connection currently cached by $\mathcal{ALG}$ both at v and w. We say that $\mathcal{ALG}$ *evicts* (v, w) *at* v if $\mathcal{ALG}$ will evict (v, w) while serving a request for a connection (v, u) for some $u \neq w$.

Theorem 5.2. *Any connection caching algorithm satisfying Property 5.1 is $(2k-1)$-competitive.*

Proof. Let $\mathcal{ALG}$ be an algorithm satisfying Property 5.1, and let $\mathcal{OPT}$ be the optimal offline connection caching algorithm. Assume that $\mathcal{OPT}$ serves each request before $\mathcal{ALG}$. We will charge misses of $\mathcal{ALG}$ to misses of $\mathcal{OPT}$ such that at most $2k-1$ misses of $\mathcal{ALG}$ are charged to each miss of $\mathcal{OPT}$.

Let m' be a miss of $\mathcal{ALG}$ generated by a request for connection (v, w). If (v, w) is ever evicted by $\mathcal{ALG}$ then let $u \in \{v, w\}$ be the endpoint at which $\mathcal{ALG}$ evicts (v, w). Otherwise let u be either v or w arbitrarily. We charge m' to the most recent miss of $\mathcal{OPT}$ on a connection incident to u (i.e. the last miss of $\mathcal{OPT}$ on a connection incident to u when serving the prefix of the request sequence up to and including r).

We shall prove that we charge at most $2k-1$ misses of $\mathcal{ALG}$ to each miss of $\mathcal{OPT}$, by proving the following two claims. Let m be a miss of $\mathcal{OPT}$ incured on request r for the connection (x, y).

Claim (1): We charge to m misses related to at most $k-1$ different connections incident with x, and at most $k-1$ different connections incident with y, other than (x, y).

Proof of Claim 1: Assume that we charge misses incured on requests to k different connections $(x, v_1), \ldots, (x, v_k)$ to m, where $v_i \neq y$ for $1 \leq i \leq k$. Then by the time $\mathcal{ALG}$ gets the k-th such miss, say on the request for connection (x, v_k), $\mathcal{OPT}$ must have had a miss incident with x following m. This is because otherwise $\mathcal{OPT}$ must contain $(x, y), (x, v_1), \ldots, (x, v_k)$ in cache(x) after serving r which contradicts the fact that cache(x) is of size k. Therefore the miss on (x, v_k) cannot be charged to m and the claim follows.

Claim (2): We charge at most one miss on each particular connection to m.

Proof of Claim 2: In order to charge a second miss on say (x, z) to m connection (x, z) must be evicted at x first (or at y in case $z = y$). By property 5.1 in order to evict (x, z), x (or y) must see at least k different connections which are also different from (x, z). This implies that $\mathcal{OPT}$ must have had a miss at x (or y) after m and before the second miss on (x, z), which is a contradiction. $\square$

A sequence given in [7] shows that the 2k-1 upper bound is tight for type 1 and type 2 $\mathcal{LRU}$ for every $k \geq 2$. It is possible to modify the sequence in [7] such that $\mathcal{LRU}$ misses every request of the modified sequence. Since $\mathcal{FIFO}$ acts exactly as type 2 $\mathcal{LRU}$ on sequences that cause type 2 $\mathcal{LRU}$ to miss on every request, and $\mathcal{FIFO}'_1$ acts as type 1 $\mathcal{LRU}$ on such sequences we obtain by the modified sequence that the 2k-1 upper bound is also tight for $\mathcal{FIFO}$ and $\mathcal{FIFO}'_1$. We omit the details of how to modify the sequence in [7] from this extended abstract.

We can extend Theorems 5.2 to obtain bounds on the competitive ratios achieved by algorithms with property 5.1 against offline connection caching algorithms that use smaller caches.

THEOREM 5.3. *Any connection caching algorithm satisfying Property 5.1 is* $(\max_{(u,v)\in E}[\frac{k(u)}{k(u)-h(u)+1} + \frac{k(v)-1}{k(v)-h(v)+1}])$*-competitive against an offline connection caching algorithm which at every vertex u of the network is only allowed to use a cache of size $h(u) \leq k(u)$.*

PROOF. The proof is an extension of the proof of Theorem 5.2. We charge misses of $\mathcal{ALG}$ to misses of $\mathcal{OPT}$ as follows. Let r be a request for connection (v, w) on which $\mathcal{ALG}$ misses. If (v, w) is ever evicted by $\mathcal{ALG}$ then let $u \in \{v, w\}$ be the endpoint at which $\mathcal{ALG}$ is going to evict (v, w). Otherwise, pick u to be either v or w arbitrarily. We charge $1/(k(u) - h(u) + 1)$ of the miss on r to each of the $k(u) - h(u) + 1$ most recent misses of $\mathcal{OPT}$ incident to u.

Let m be a miss of $\mathcal{OPT}$ incured on request r for the connection (x, y). We rephrase Claim 1 and Claim 2 as follows.

Claim (1): We charge to m misses related to at most $k(x) - 1$ different connections incident with x, and at most $k(y) - 1$ different connections incident with y, other than (x, y).

Claim (2): We charge a fraction of at most one miss on each particular connection to m.

It is easy to check that the theorem follows from these two claims. $\square$

In the context of a single-cache, Young [16] defined a paging algorithm ALG as *conservative* if it satisfies the following property.

PROPERTY 5.4. *On any consecutive input subsequence containing k or fewer distinct page references,* ALG *will incur k or fewer misses.*

Young observed that LRU, FIFO and other natural policies are conservative, and that being conservative suffices to guarantee a competitive ratio of k.

Consider a type 1 or a type 2 connection caching algorithm derived from a conservative single-cache algorithm. There may be more than k misses on requests for connections with v generated by a subsequence that includes requests for no more than k different connections with v. So Property 5.4 above may not hold from the perspective of a single vertex. This is due to dumps that occur at v. Therefore, in the context of connection caching we prefer to count evictions rather than misses (Note that misses and evictions are equivalent in the single-cache setting), and we define *conservative connection caching* algorithm as follows.

DEFINITION 5.5. *A connection caching algorithm is conservative if any subsequence of requests that contains only k distinct requests touching u, causes at most k evictions at u.*

Using this definition we have that any type 1 connection caching algorithm derived from a conservative single-cache policy is a conservative connection caching policy. Furthermore it is easy to see that any algorithm satisfying property 5.1 is a conservative connection caching algorithm. For conservative connection caching algorithms we can prove the following.

THEOREM 5.6. *Any conservative connection caching algorithm is* $(2k)$*-competitive*

PROOF. We charge evictions of $\mathcal{ALG}$ to misses of $\mathcal{OPT}$ such that at most $2k$ eviction of $\mathcal{ALG}$ are charged to each miss of $\mathcal{OPT}$. The theorem then follows using Lemma 3.1.

We charge an eviction of $\mathcal{ALG}$ at u by a request r to the most recent miss of $\mathcal{OPT}$ on the prefix of the sequence of requests up to and including r on a connection with u. By the definition of a conservative connection caching algorithm a miss on a connection (x, y) is charged by at most k eviction at x and k evictions at y, giving the theorem. $\square$

We also have the following generalization.

THEOREM 5.7. *Any conservative connection caching algorithm is* $(\max_{(u,v)\in E}[\frac{k(u)}{k(u)-h(u)+1} + \frac{k(v)}{k(v)-h(v)+1}])$*-competitive against an offline connection caching algorithm which at every vertex u of the network is only allowed to use a cache of size $h(u) \leq k(u)$.*

6. CONCLUSION AND OPEN PROBLEMS

The underlying basic open questions are to further quantify the power of communication. Can the types be separated with appropriate upper and lower bounds?

We showed that all type 3 deterministic marking algorithms are k-competitive. One algorithm in this class, $\mathcal{FWF}$, is also a type 2 algorithm but not a demand caching algorithm. The best upper-bound we know of for the competitive-ratio of any type 1 and demand type 2 algorithms is $2k - 1$ (tight for $\mathcal{LRU}_1$ and $\mathcal{LRU}_2$). Are there type 1 or demand-caching type 2 algorithms with better competitiveness? If so, are there type 1 or type 2 marking or conservative algorithms that achieve better performance?

Similar questions arise for randomized algorithms. The simple randomized marking algorithm (MARK) [8] has competitive ratio of $2H_k - 1 = O(\log k)$ [1]. Fiat et al [8] gave an H_k lower bound on the competitiveness of any single-cache randomized algorithm. This lower bound carries over for randomized connection caching algorithms. McGeoch and Sleator closed the gap for single-cache algorithms by introducing a randomized algorithm with competitive ratio of H_k [11]. Their algorithm combined with a result in [7] yields a $2H_k$-competitive randomized type 1 connection caching algorithm. With further communication, we can only show weaker bounds: $\mathcal{MARK}_2$ is $4H_k$-competitive, and that $\mathcal{MARK}_3$ is $\Omega(k)$-competitive. Can the extra power of communication be exploited to yield better, perhaps even H_k-competitive, randomized connection-caching algorithms?

7. REFERENCES

[1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. In *Proceedings of the 4th European Symposium on Algorithms*, pages 419–430. Springer-Verlag, Lecture Notes in Computer Science Vol. 1136, 1996.

[2] S. Albers. Private Communication, May 1999.

[3] S. Albers. Generalized connection caching. Manuscript, 1999.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. RFC 1945, MIT/LCS, May 1996. `http://ds.internic.net/rfc/rfc1945.txt`.

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

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

[7] E. Cohen, H. Kaplan, and U. Zwick. Connection caching. In *Proc. 31st Annual ACM Symposium on Theory of Computing*. ACM, 1999.

[8] A. Fiat, R. Karp, M. Luby, L. McGoech, D. D. Sleator, and N. Young. On competitive algorithms for paging problems. *J. Alg.*, 12:685–699, 1991.

[9] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2068, MIT/LCS, May 1997. `ftp://ds.internic.net/rfc/rfc2068.txt`.

[10] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):70–119, 1988.

[11] L. McGoech and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

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

[13] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[14] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, Reading, MA, 1994.

[15] N. Young. On line file caching. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1998.

[16] N. E. Young. *Competitive paging and Dual-Guided On-Line Weighted Caching and matching algorithms.* PhD thesis, Princeton University, 1991.