Connection Caching under Various Models of Communication

Edith Cohen

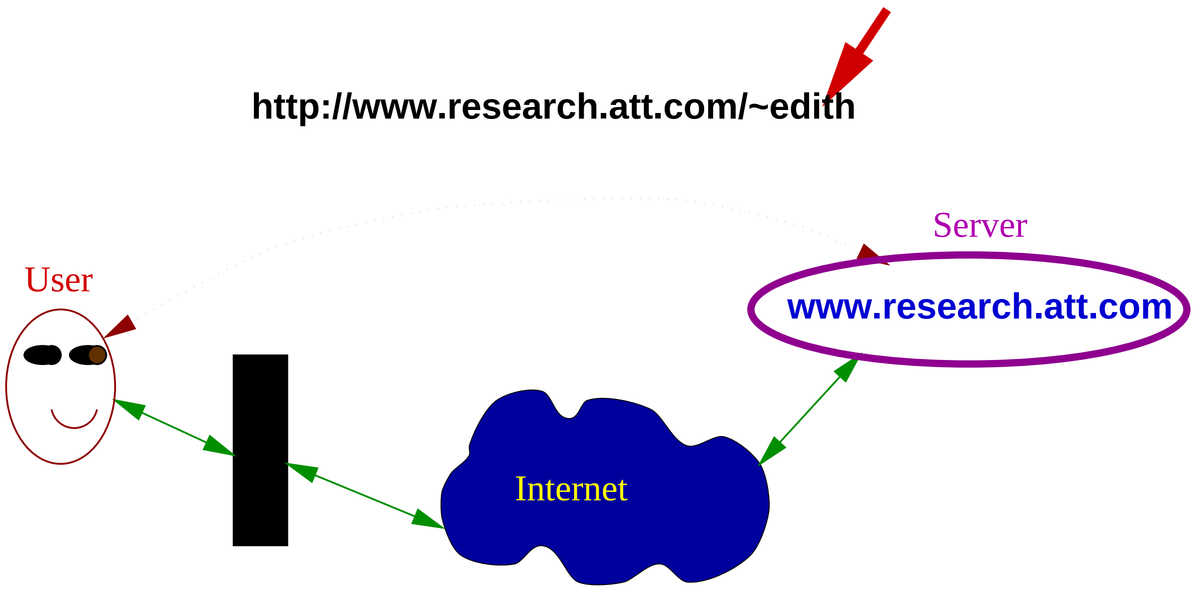
AT&T Labs – Research

Joint with: Haim Kaplan and Uri Zwick
Tel-Aviv University

http://www.research.att.com/~edith

edith@research.att.com

What happens when you click ?



- TCP connection is established (can be reused) to the HTTP server `www.research.att.com`
- HTTP request `GET /~edith` sent to server. Server transmits response back.

HTTP over TCP

TCP (Transmission Control Protocol):

- 3-way hadshake to establish/terminate connection
- Implemented over connectionless IP: connection state exists only at end-points.
- No HTTP requests $\Rightarrow$ no packets

HTTP/1.1: connection re-use

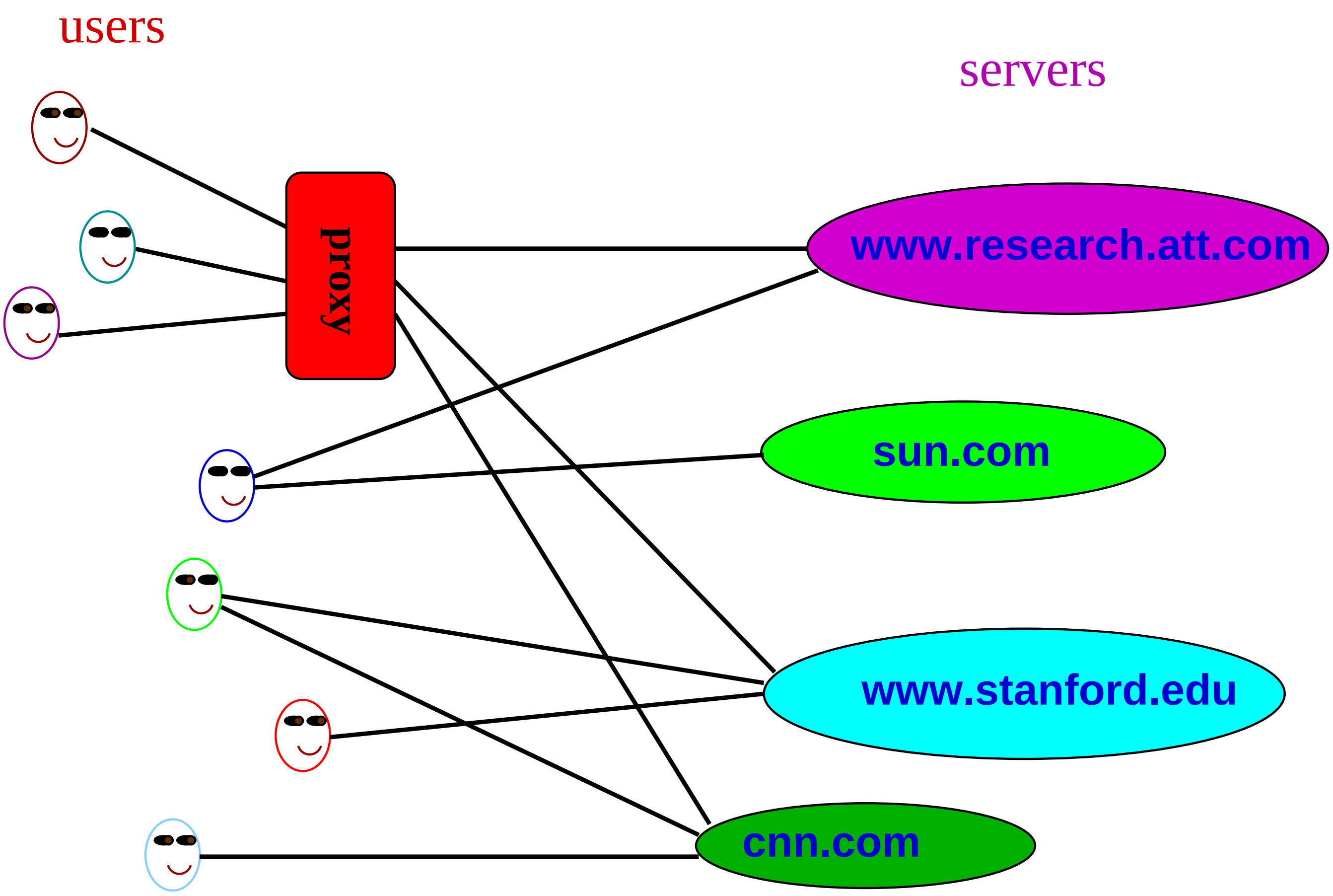
- A connection should be kept idle, serving future requests, until explicitly closed.
- Either endpoint may terminate connection

Tradeoffs:

◇ Minimize number of connection establishments (latency and overhead).

◇ subject to: Limits on number of open connections hosts can maintain.

Global Connections Picture

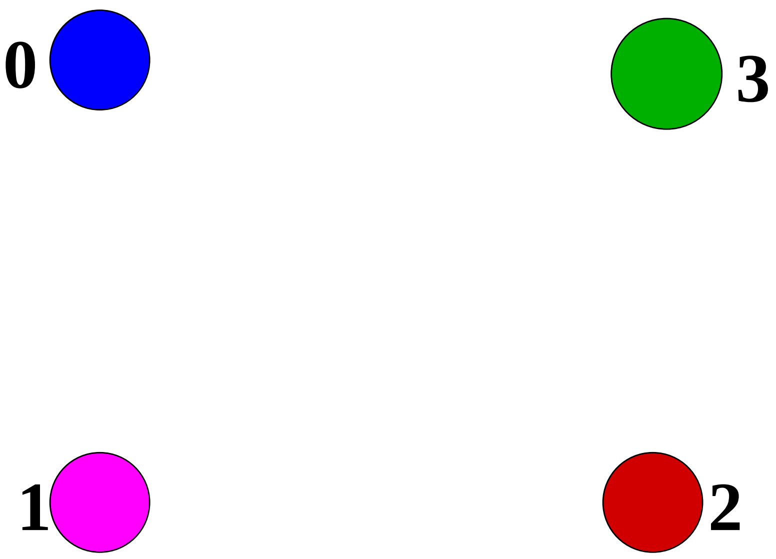


Connection Caching Model

INPUT:

1) Set of nodes V (hosts) with capacities $k(v)$ $v \in V$ (max # of simultaneous open connections v can keep).

2) Sequence of edges from $V \times V$ (HTTP requests).



$\forall v, k(v) = 2$

(0,2) (2,3) (0,3) (1,3) (1,2) (2,3)

Servicing Requests

The caches state is captured by a subset $E \subset V \times V$ such that $\mathrm{degree}_E(v) \leq k(v)$.

Serve request (v, w) as follows:

- If $(v, w) \in E$ do nothing. (hit)
- Otherwise, (miss)

  a) if $\mathrm{degree}(v) = k(v)$
     delete an edge incident to v.

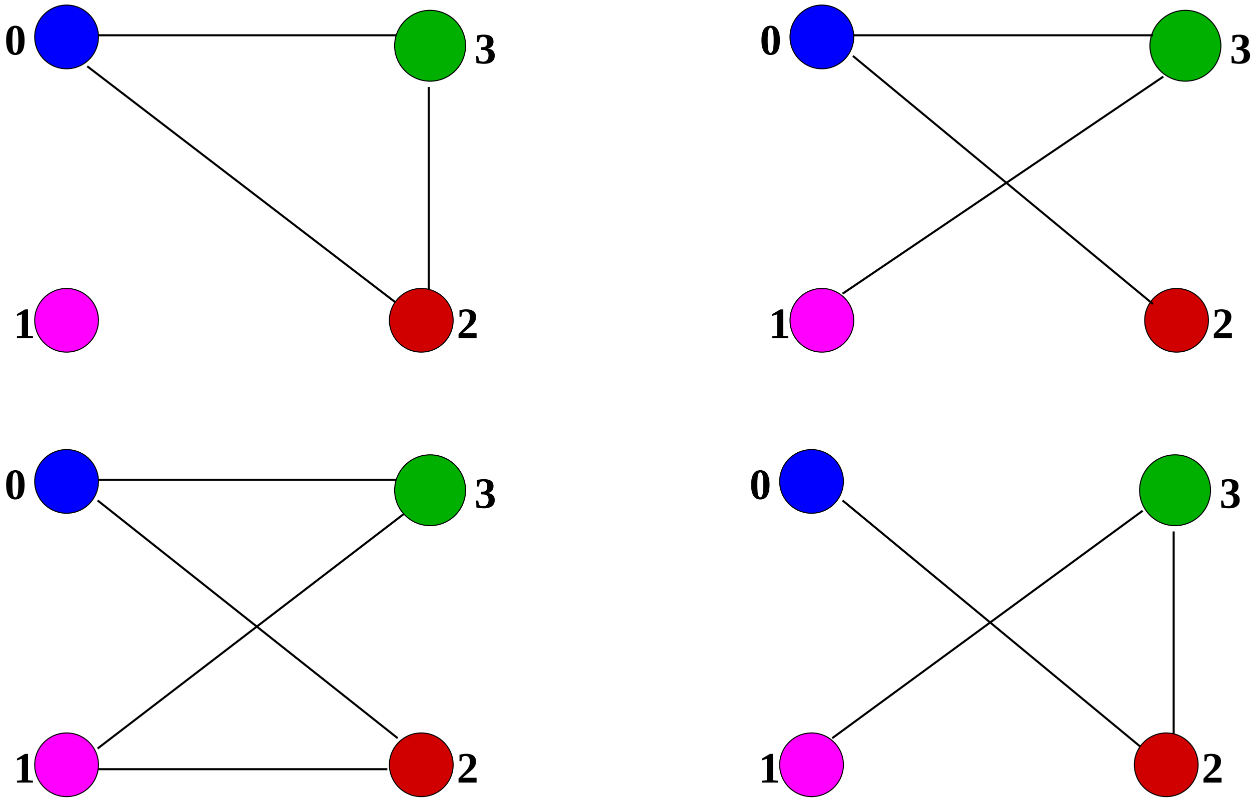
  b) if $\mathrm{degree}(w) = k(w)$
     delete an edge incident to w.

  c) add (v, w) to E.

Goal: Minimize the number of misses.

Example

(0,2) (2,3) (0,3) (1,3) (1,2) (2,3)

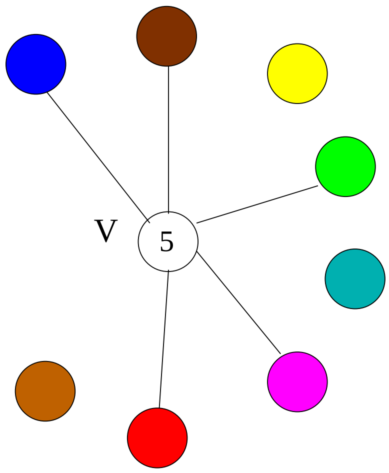


hit miss evict dump hole

Connection Caching generalizes Caching

- cache of size k, n pages
- request sequence of pages
- to serve a request, page must be in cache

Connection caching equivalent:



Communication Models

Range between Distributed and Centralized

type 1

- runs locally at each host
- sees only local requests
- dumps: v does not know if w evicts (v, w)

---

type 2

- runs locally at each node
- sees only local requests
- dumps: v knows when w evicts (v, w)

---

type 3

- runs locally at each node
- sees only local requests
- dumps: v knows when w evicts (v, w)
- hosts v, w may communicate 1 extra bit per request (v, w)

Connection Caching Algorithms

- Offline / Online
- Disributed . . . Centralized
- Deterministic / Randomized

Paging Algorithm $\mathrm{A} \Rightarrow$ Connection Caching algorithms $\mathcal{A}^*, \mathcal{A}$

$\mathcal{A}^*$ is type 1 - implements A at each node, ignores dumps.

$\mathcal{A}$ is type 2 - fills holes before evicting.

$\mathrm{LFD} \Rightarrow \mathcal{LFD}^*, \mathcal{LFD}$

$\mathrm{LRU} \Rightarrow \mathcal{LRU}^*, \mathcal{LRU}$

Competitive Ratio

[ST85]

An online paging algorithm A is c-competitive if on any sequence of requests

$$\frac{\mathrm{miss}(\mathrm{A})}{\mathrm{miss}(\mathrm{OPT})} \leq c \quad (+\ \text{const.})$$

where OPT is the optimal offline algorithm.

Bounds

Theorem: [CKZ99]
If A is c-competitive then $\mathcal{A}^*$ is $(2c)$-competitive.

Corollaries:

- $\mathcal{LFD}^*$ is a 2-approximation algorithm (tight!)
- For any marking algorithm A, $\mathcal{A}^*$ is $(2k)$-competitive (using [FKLMSY91]).
  - $\mathcal{LRU}^*$ is $(2k)$-competitive
  - $\mathcal{FWF}^*$ is $(2k)$-competitive
- For any marking algorithm A, $\mathcal{A}^*$ is
$$\frac{2k}{(k-h+1)}\text{-competitive}$$
  against offline optimal algorithm with caches of size h (using [FKLMSY91]).
- There is a randomized connection caching algorithm that is $(2H_k)$-competitive (using [ACN96])

Single-Cache Marking Algorithms

Maintain a mark bit for each page in cache:

- Pages enter the cache marked.
- Only unmarked pages can be evicted.
- When all pages are marked and we need to evict, all cached pages are unmarked.

Theorem: [FKLMSY 91] Marking algorithms are

- k-competitive
- $\frac{k}{k-h+1}$-competitive against an offline algorithm with cache of size h.

- LRU is a marking algorithm.
- FWF (Flush When Full) – weakest marking algorithm.

Connection-Caching Marking Algorithms

type 1:

run a marking algorithm at each node, ignoring dumps. e.g. $\mathcal{LRU}^*$, $\mathcal{FWF}^*$

---

type 2:

run a marking algorithm at each node. Fill all holes before unmarking. e.g. $\mathcal{LRU}$, $\mathcal{FWF}$

---

type 3:

mark bit is shared. Either side may unmark

Lemma: $\mathcal{LRU} \preceq \mathcal{LRU}^* \preceq \mathcal{FWF}^*$

Lemma: $\mathcal{LRU}$ is $\geq (2k-1)$-competitive

Corollary: $\mathcal{LRU}$ and $\mathcal{FWF}^*$ are
$\geq (2k-1)$-competitive

competitiveness bounds for marking algorithms

	upper	lower
type 1	$2k-1$	$(2k-1)$ for $\mathcal{LRU}^*$,$\mathcal{FWF}^*$
type 2	$2k-1$	$(2k-1)$ for $\mathcal{LRU}$
$*$ type 3	k	tight for all! (single-cache LB)

Observation: $\mathcal{FWF} \in$ type 2 $\cap$ type 3

k, h upper bounds for marking algorithms

Algorithm uses caches of size $k(v)$, offline optimal uses caches of size $h(v)$.

Upper bound for type 1 and type 2 marking algorithms:

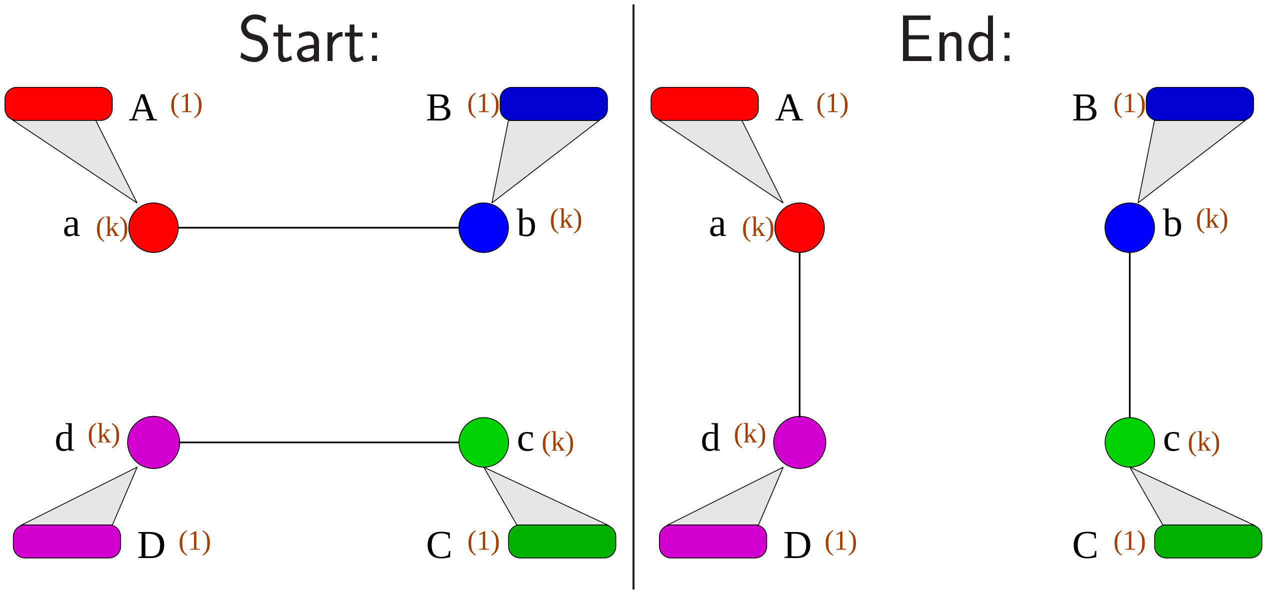
$$\max_{(u,v)\in V\times V} \frac{k(u)}{k(u)-h(u)+1} + \frac{k(v)}{k(v)-h(v)+1}$$

Upper bound for type 3 marking algorithms:

$$\max_{v\in V} \frac{k(v)}{k(v)-h(v)+1}$$

$2k-1$ lower bound for competitiveness of $\mathcal{LRU}$

$A = \{A_1, \ldots, A_{k-1}\}$ $B = \{B_1, \ldots, B_{k-1}\}$ $C = \{C_1, \ldots, C_{k-1}\}$ $D = \{D_1, \ldots, D_{k-1}\}$



requests	$\mathcal{OPT}$	$\mathcal{LRU}$
$(a,d),(b,c)$	2m 2e: (a,b) (c,d)	2m 4e: (a,A_1) (b,B_1) (c,C_1) (d,D_1)
$i=1,\ldots,k-2$: (a,A_i) (b,B_i) (c,C_i) (d,D_i)	4h	4m 4e: (a,A_{i+1}) (b,B_{i+1}) (c,C_{i+1}) (d,D_{i+1})
(a,A_{k-1}) (b,B_{k-1}) (c,C_{k-1}) (d,D_{k-1})	4h	4m 2e: $(a,b)(c,d)$

k upper bound for type 3 marking

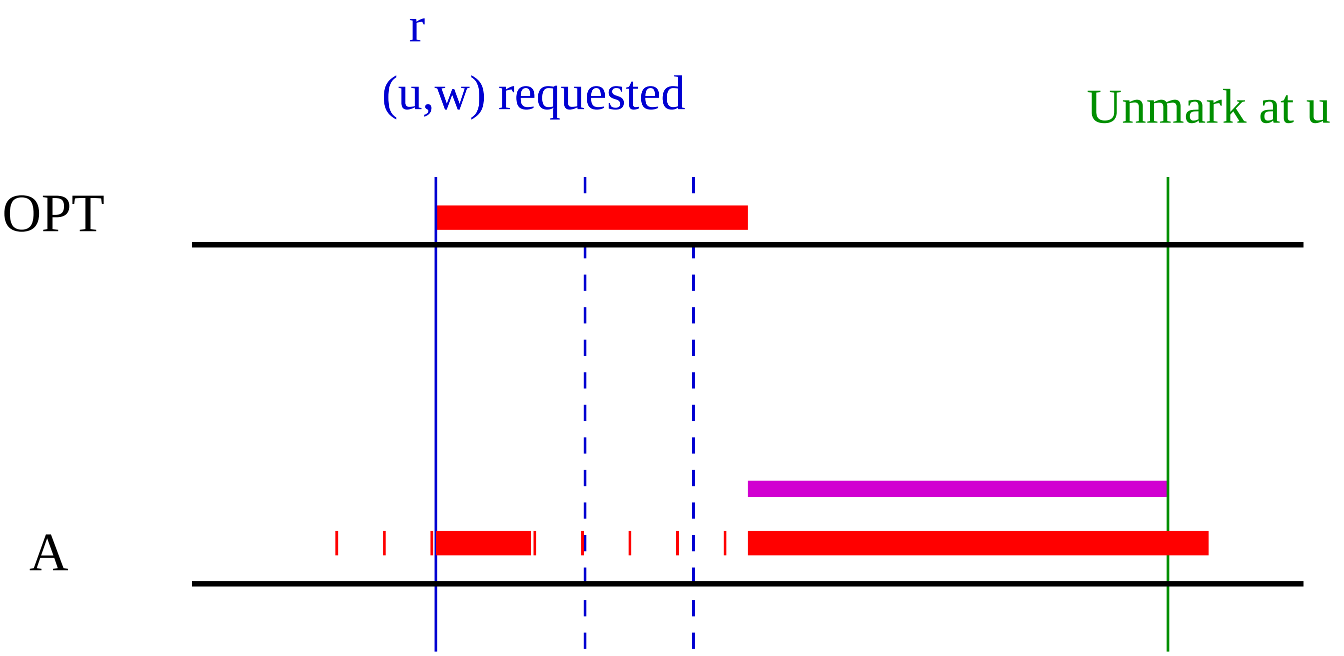
- We charge each unmark step to a miss of $\mathcal{OPT}$

Let (u, v) be a request that caused unmark at u.
Consider the time after $\mathcal{OPT}$ and before $\mathcal{A}$ serves the request:
Let $(u, w) \in \text{cache}(\mathcal{A}(u)) \setminus \text{cache}(\mathcal{OPT}(u))$
The unmark is charged to the most-recent miss r incurred when $\mathcal{OPT}$ inserted (u, w) into its cache.

Lemma: r is charged at most once.

- r can not be charged simultaneously for unmarks at u, w:
unmark at u occurred because of $v \neq w$.
- r can not be charged more than once at different times:
(u, w) must be marked when r is charged.
(u, w) is unmarked after the first charge of r.
if r is charged again, (u, w) must have been re-marked $\Rightarrow$ a request r' happened between 1st and 2nd charge $\Rightarrow$ 2nd charge is placed on $r' \neq r$.

k upper-bound proof



(u,w) in cache

(u,w) possibly in cache

(u,w) must be marked

Communication and Randomized Marking

Randomized marking algorithm:
evict an unmarked page chosen uniformly at random [MS91].

- Communication does not seem to help randomized connection-caching marking

type 1: $(4H_k - 2)$-competitive
(from $(2H_k - 1)$-competitiveness of single-cache algorithm [MS91]).

type 2: $4H_k$-competitive (perhaps better)

type 3: $\Omega(k)$-competitive (!!!)

Open Questions

- ? Is there an offline $(2-\varepsilon)$- approximation algorithm?
- ? Is there a k-competitive distributed demand connection caching algorithm that uses no additional communication (type 2) ?
- ? Is $(2k-1)$-competitiveness the best that can be achieved distributively with no notification of connection closures ?
- ? Is there a randomized H_k-competitive connection caching algorithm ?