Prioritized Token-Based Mutual Exclusion for Distributed Systems

Frank Mueller

Institut für Informatik
Humboldt-Universität zu Berlin
10099 Berlin (Germany)

e-mail:
mueller@informatik.hu-berlin.de

WWW:
`http://www.informatik.hu-berlin.de/~mueller`

Overview

- motivation, assumptions
- unprioritized protocol
- priority support, step-by-step
- message overhead
- measurements

Motivation

- distributed mutual exclusion
- fast protocol
- support for priorities $\Rightarrow$ real-time
- known worst-case and best-case behavior
- asynchronous

$\Rightarrow$

- implemented in DSM-Threads
- POSIX.1c-like API

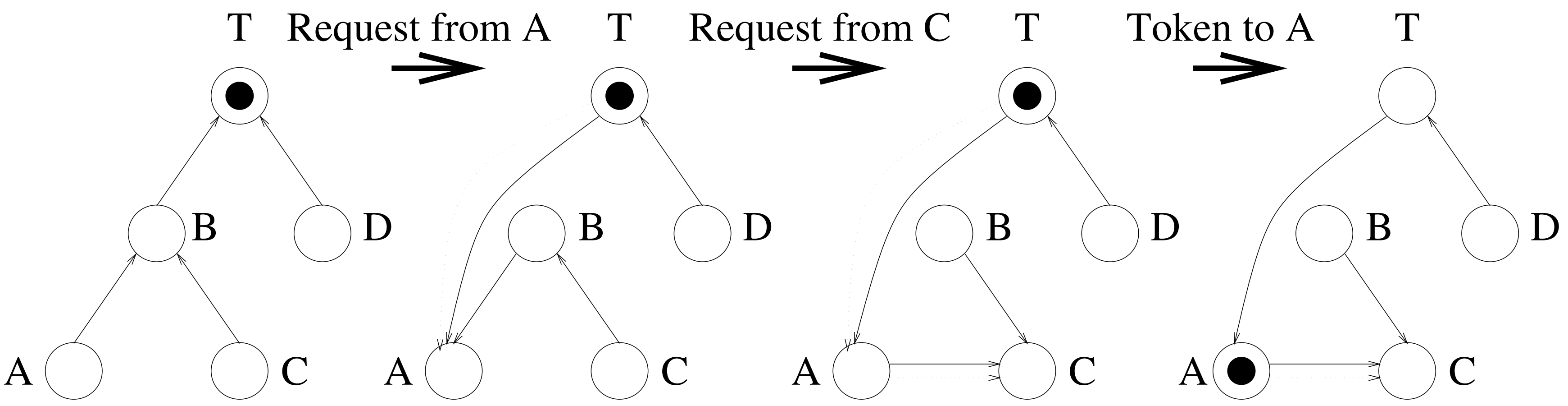
Assumptions

- fully connected network
- reliable message passing
- out-of-order delivery OK
- t(local operations) $\ll$ t(msg passing)
- $\Rightarrow$ reduce $|$msgs$|$

Unprioritized Protocol

- by Naimi *et. al*
- nodes form a logical tree
- root owns token (mutex)
- arcs point towards probable owner (last requester)
- next points towards next requester (distr. queue)
- at most one request per node at a time
- k requests in parallel $\Rightarrow$ k+1 trees in forest
- after requests arrive $\Rightarrow$ 1 tree, root = last req.
- avg. O(log(n)) msgs
- worst case O(n) msgs

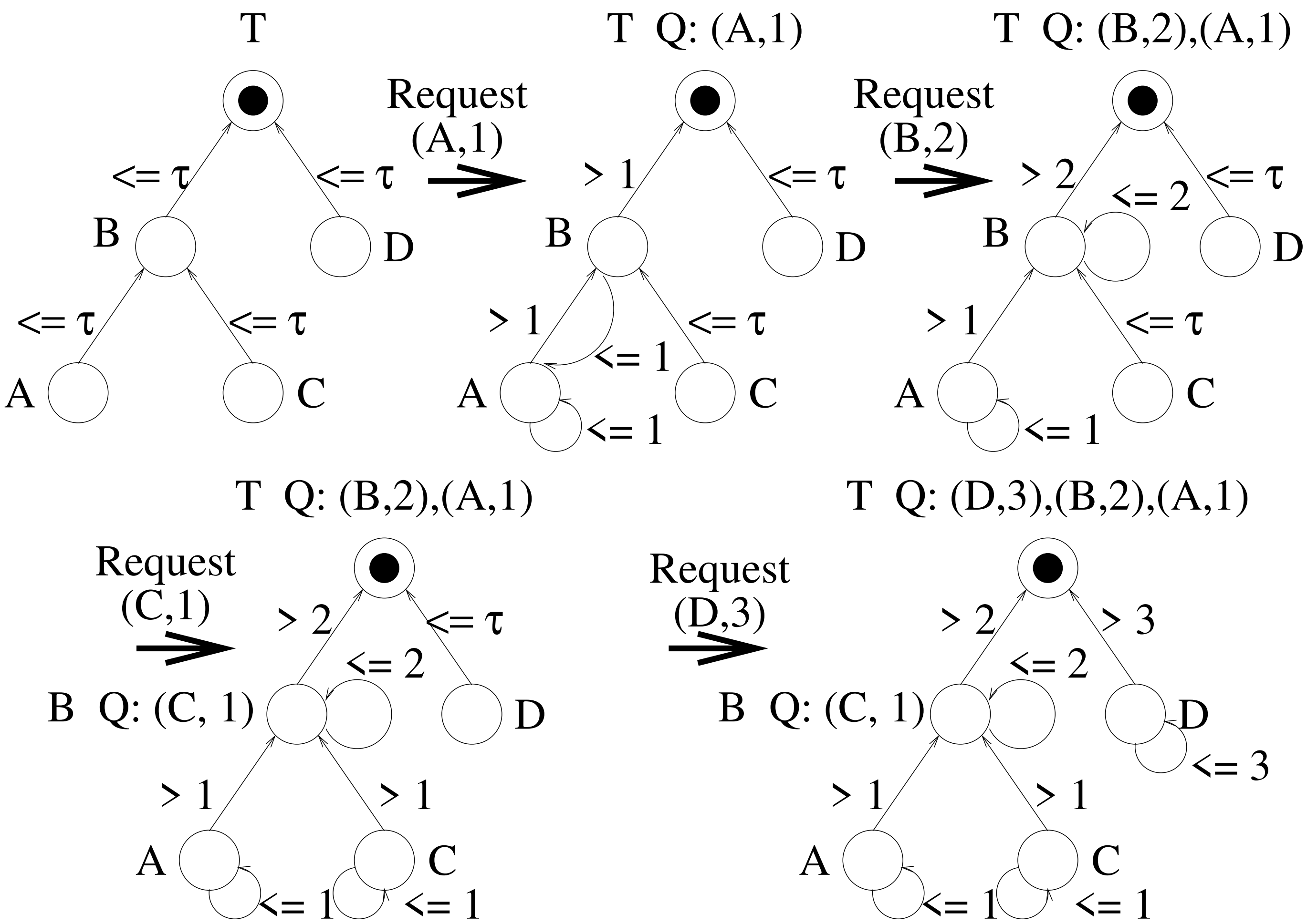
Unprioritized Example



Priority Support

- assume: higher prio = more important
- order requests by prio (+ FIFO at each prio level)
- idea: accumulate prio info at nodes during request forwarding
- $\Rightarrow$ arcs tagged with priorities
- if node I receives req. from N at prio 10 and passes req. on, then create arc I $\rightarrow$ N w/ tag ≤ 10
- problem: race, cannot insert req. at any place in distr. Q
- solution: local queues

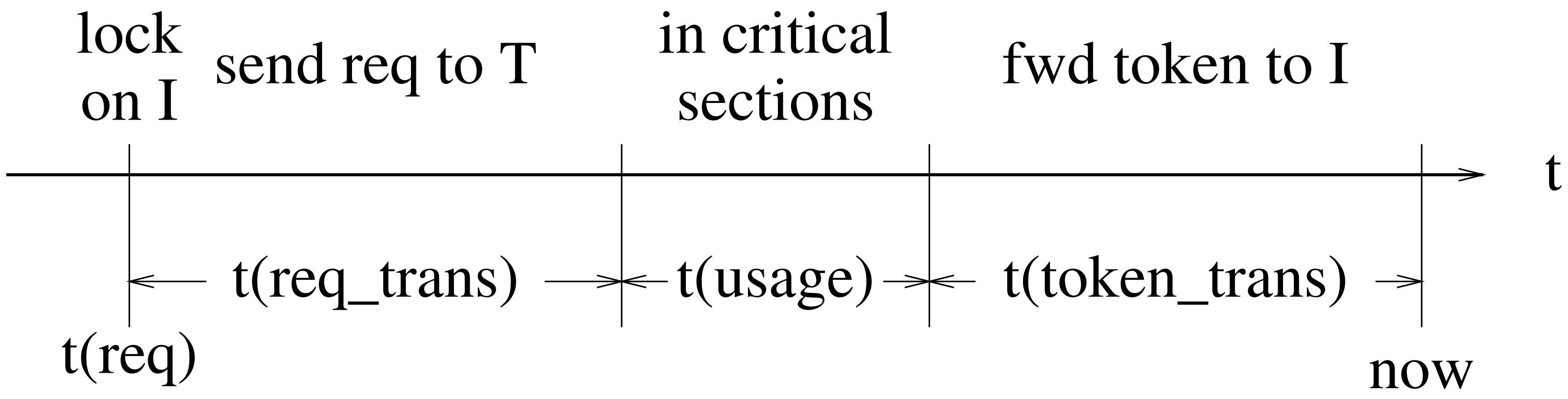
Example of Token Requests



Local Queues

- queue request locally if node issued request $\geq$ prio
- problem: FIFO within one prio level (who was first?)
- inefficient solution: global logical clock, timestamps
- better solution: use local time facilities, estimate remote times
- accumulate token usage times
- log request time $\Rightarrow$ known latency upon token reception
- approximate usage time of other requests:
  $t_X(new\_usage) = t_X(usage) + (now - t_X(receipt)) * \frac{t_I(usage)}{t_I(latency)}$
- $\Rightarrow$ order FIFO via usage times within prio level

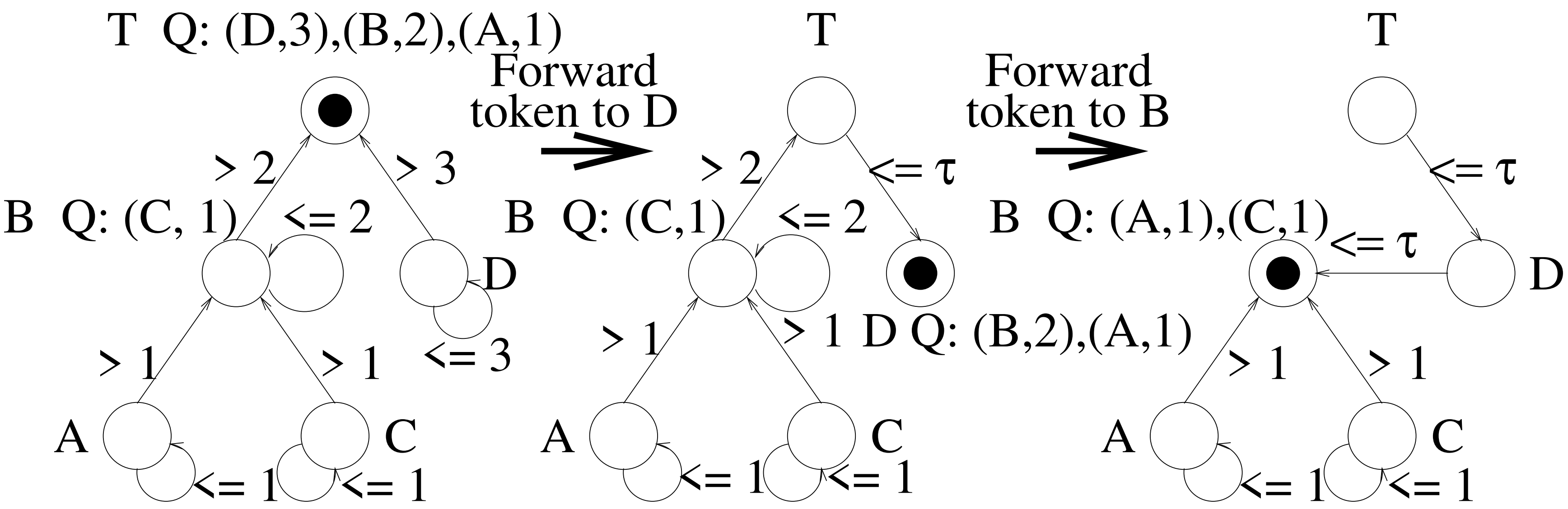
Events and Times for Token Requests



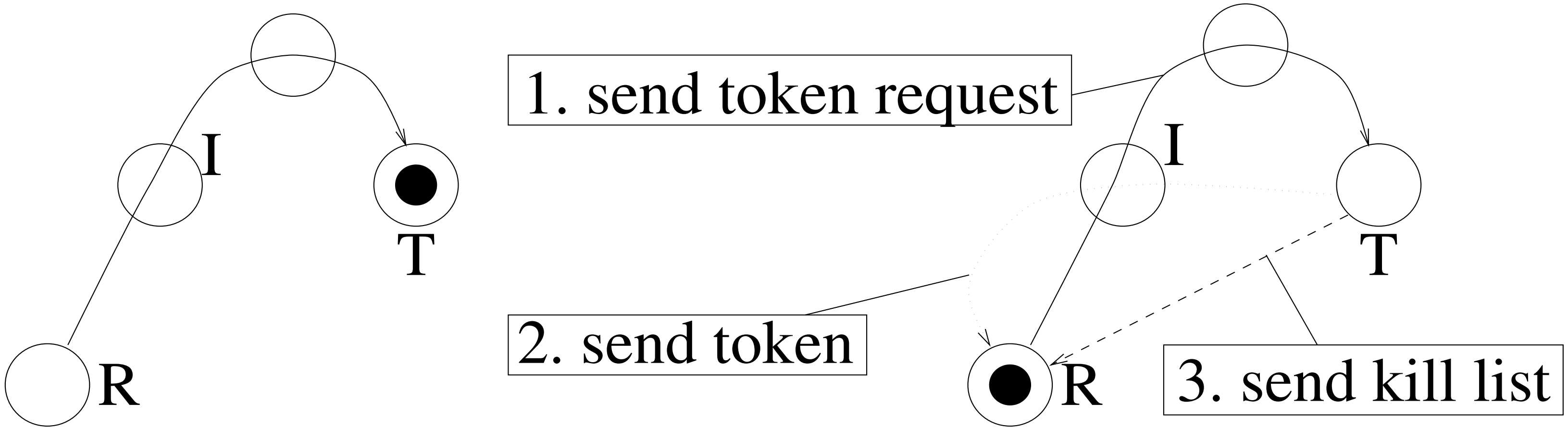
Token Forwarding

- token used $\Rightarrow$ send to head(Q), piggyback Q
- set probable owner to new token holder (for any prio)
- problem: higher prio request may trail lower one and and may pass initiator of lower one (trailing)
- bad effect: $|hops| > n$ for requests
- solution: log forwarded requests locally, add kill list
- kill list collects nodes along propagation path
- stop request at T if probable owner I in kill list (loop)
- token automatically sent from I to requester R
- kill list sent from T to requester R
- kill list indicates requests to be removed from local Qs
- kill lists also piggybacked upon token forwarding

Example of Token Forwarding



Abstract Model of Token Forwarding



(a) Request Reaches Token (b) Request Trails Token

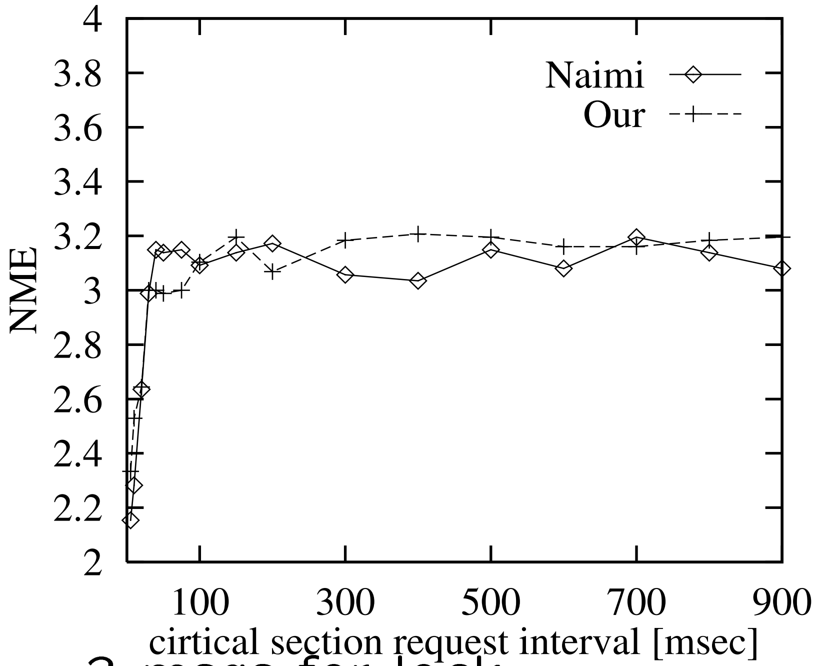
Message Overhead

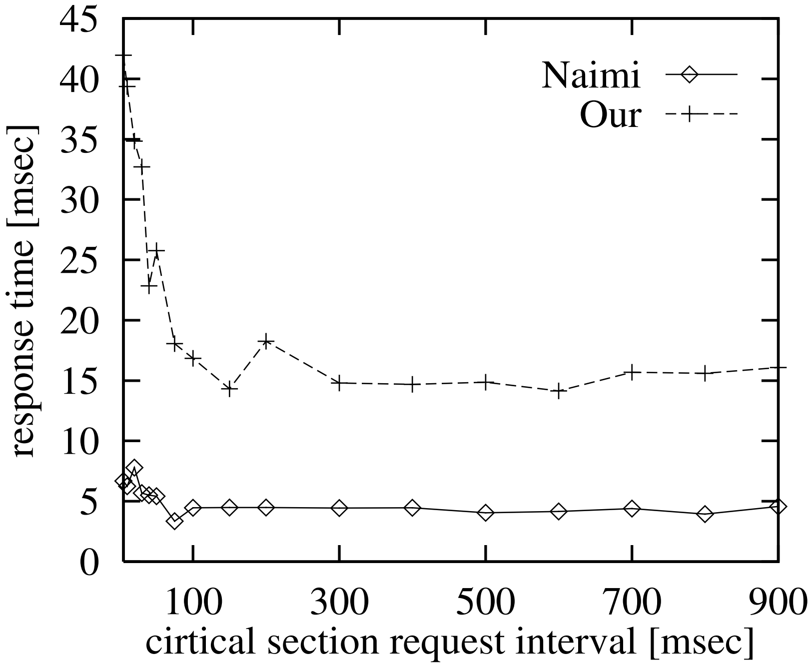
- avg. O(log(n)) due to logical tree (see Naimi et. al.)
- even for 1 prio level, better than Naimi (local Qs save hops)
- worst case: n-1 forwarded requests, 1 token fwd, 1 send kill list
- correctness: see tech report
- 2 requests handled in same order as issued? $\Rightarrow$
- timeliness: *relatively fair* if δ between 2 requests $> \omega(n-1)$ where $\omega(k) =$ max. comm. overhead for k msgs

Measurements

- DSM-Threads (POSIX-like Threads w/ DSM)
- our protocol *vs.* asynchronous version of Naimi's
- measures:
  - NME: number of messages exchanged
  - response time: time between request and getting token
- 8 tasks, 10 locks
- measurement series w/ locks issued 5-900 ms apart
- at each measurement point random distribution ± 5 ms
- 3 experiments (simulate different networks):
  - 8 threads, 1 process, SPARC 60 MHz, shared-mem (high)
  - 8 processes, SMP 2x SPARC 60MHz, TCP (medium-speed)
  - 8 processes, 8 SPARCs 40-80 MHz, TCP (low-speed)
- 10-13 ms for ping (incl. overhead DSM-Threads layer)

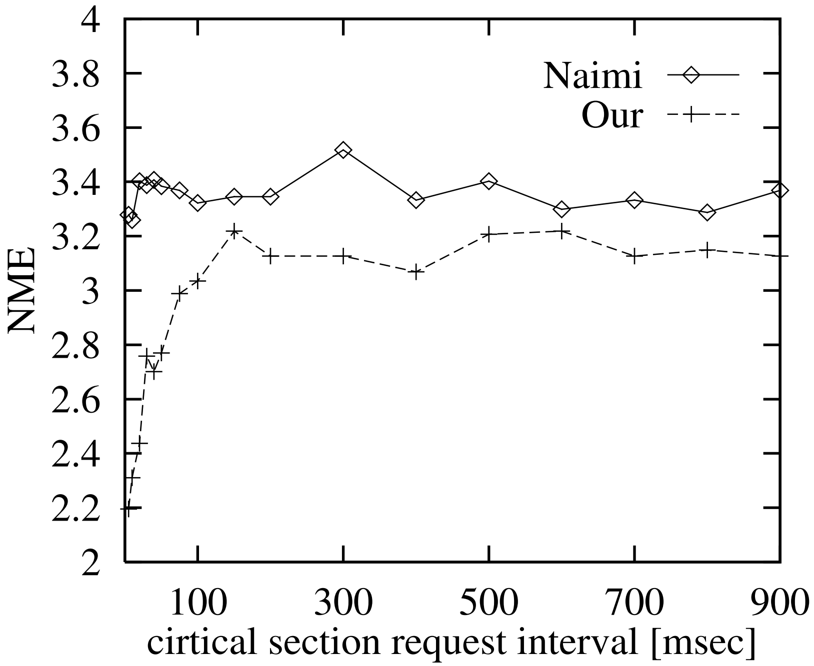
Simulated Comm., via Physically Shared Memory

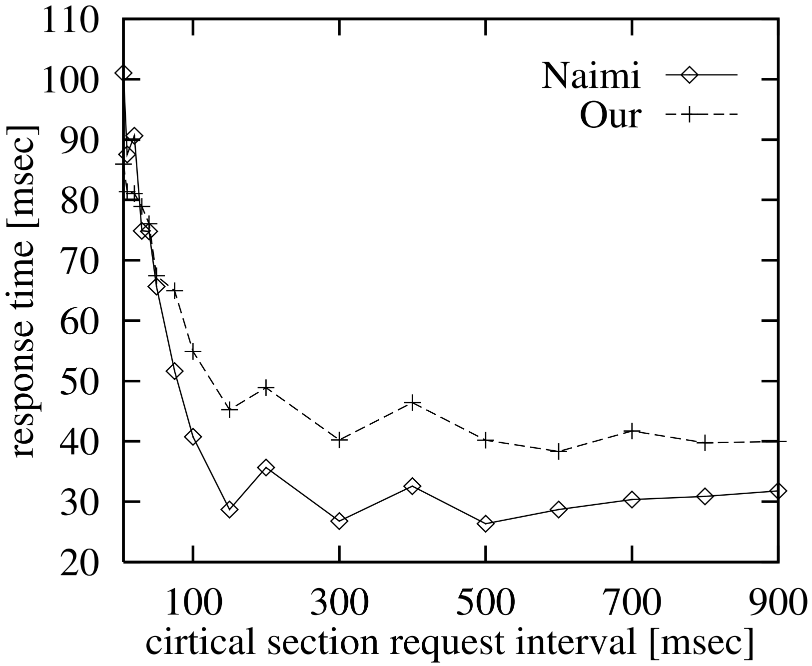




- 3 msgs for lock
- short intervals:
  - requests arrive faster than tokens are passed
  - $\Rightarrow$ chain of requesters $\Rightarrow$ 2 msgs
- response time: Naimi $= \frac{1}{3}$ ours protocol
- $\Rightarrow$ computational overhead of our protocol (larger msgs, more functionality)

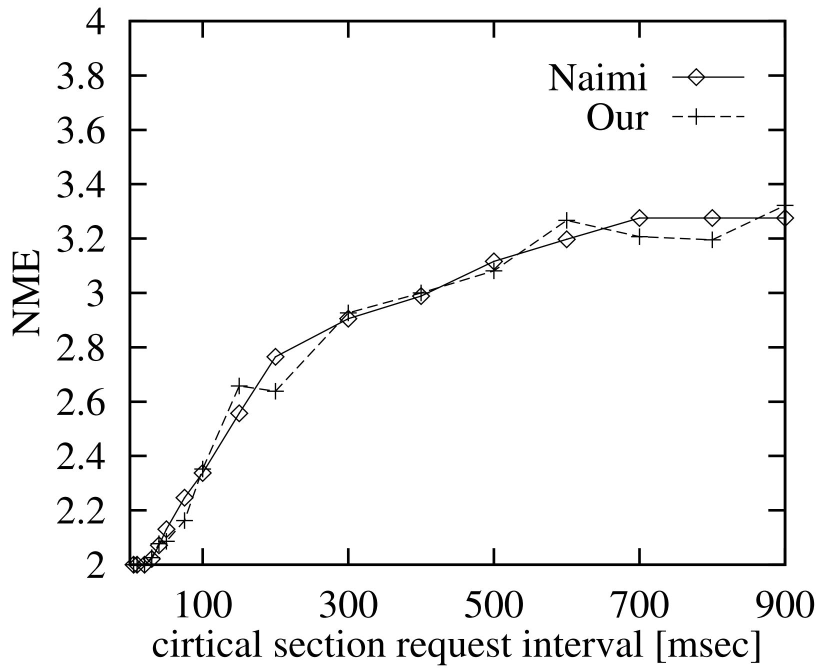
TCP Comm., SMP with 2 Processors

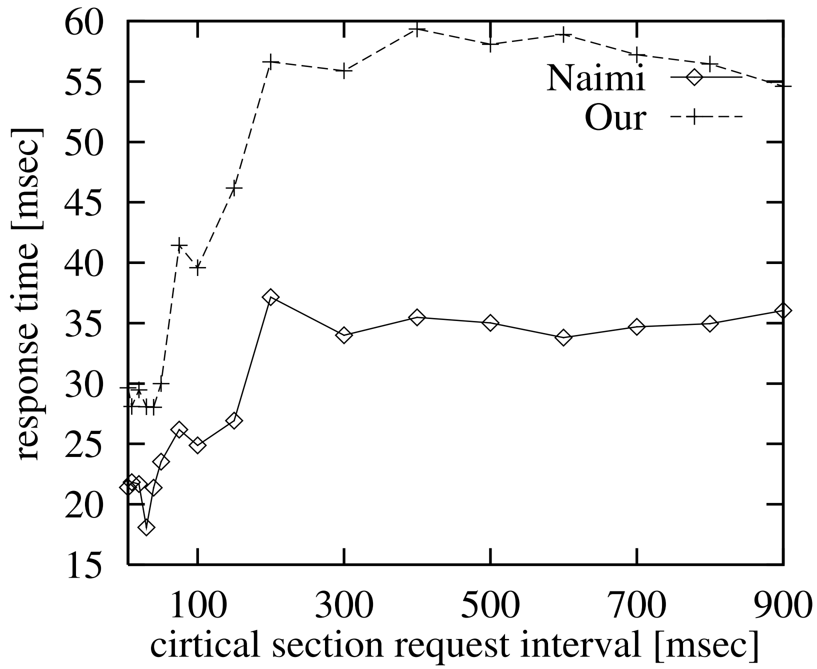




- NME(our approach) $<$ NME(Naimi) $\Rightarrow$ local Qs
- response time: Naimi $= \frac{3}{4}$ ours protocol
  $\Rightarrow$ fewer msgs
- less contention (larger intervals) $\Rightarrow$ shorter response time

TCP Comm., Distributed (Different Workstations)





- NMEs about same
- response time: Naimi $= \frac{2}{3}$ ours protocol (w/ 3 msgs)
- small intervals: many local locks (dismissed)
  $\Rightarrow$ smaller sample size (maybe misleading)

Related Work

- Goscinski: distr. lock w/ broadcast + prio
- Chang: prio extensions for
  - various broadcast protocols
  - fixed logical trees w/ token
- protocols w/ global logical clocks + timestamps: $\Rightarrow$ can extend to prio
- problem: all these have NME $>$ O(log(n))
- exceptions w/ NME $=$ O(log(n)):
  - Raymond: fixed logical tree, send token O(log(n)), no prio
  - Naimi et. al.: dynamic logical tree, send token O(1), no prio
  - our protocol: dynamic logical forest, send token O(1), prio
- all above: synchronous (ours: asynchronous)
- all above: one request per node (ours: multiple)

Conclusion

- new **prioritized** distributed mutex protocol
- avg. msgs O(log(n)), worst O(n) (same as best non-prio protocol)
- token-passing scheme
- local queues w/ requests
- requests order by prio (FIFO within each prio level)
- set of logical trees w/ prios on edges to guide requests
- forwarding of queues, local bookkeeping (time)
- supports out-of-order msg delivery + asynchronous msg handling
- can use local queues in non-prio protocols
- 30% longer response times (than non-prio)
- suitable for real-time and POSIX-compliant environments