Comparing and Evaluating epoll, select and poll Event Mechanisms

Louay Gammo, Tim Brecht,

Amol Shukla, and David Pariag

University of Waterloo

`http://cs.uwaterloo.ca/~brecht`

Outline

- Event-driven Internet server design and the userver
- select, poll and epoll overview
  - trying to reduce epoll_ctl overhead
- Performance Comparisons
  - 1 byte workload
  - Static SPECweb99-like workload
- Discussion

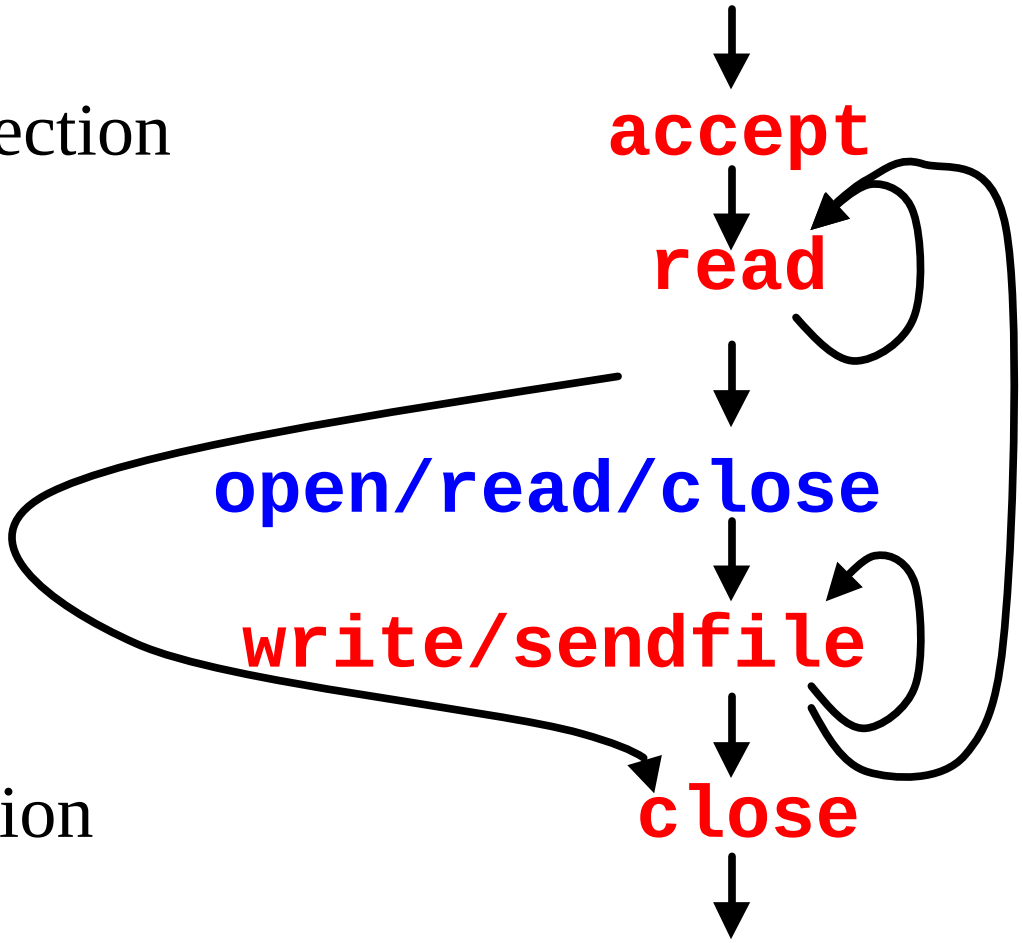
userver

- user-level, event-driven, open source web server
  - performance close to TUX [USENIX 2004]
- support for multiple event mechanisms
  - select, poll, epoll, SEND
- designed for experimentation
  - statistics, debugging and tracing facilities
- uses Linux features
  - e.g., sendfile with TCP_CORK / UNCORK

http://www.hpl.hp.com/research/linux/userver

A Typical Connection 1.1

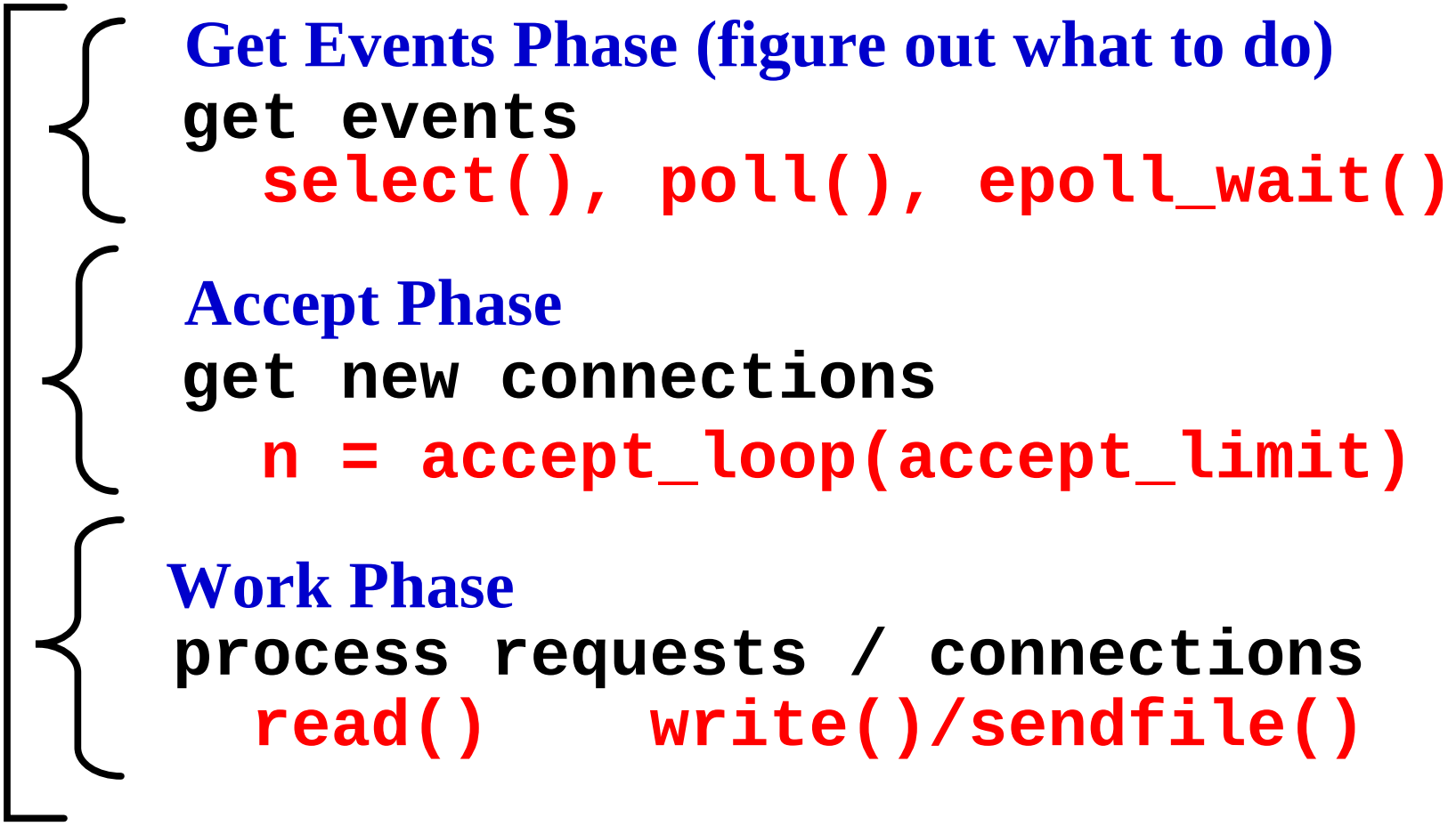
- server gets new connection
- server reads request
  - parse request
  - generate reply
- server sends reply
- server closes connection



Event Mechanisms

- select, poll
- explicit event delivery [Banga et al 1999]
- /dev/poll, [Solaris, Provo & Lever 2000]
- SEND Kernel [Ostrowski 2000]
- kqueue [Lemon 2001]
- signal per fd [Chandra & Mosberger 2001]
- /dev/epoll & epoll [Lebenzi, Weekly, Nagar et al.]
- Good source of info C10K Page [Dan Kegel]

Phases in the Event-Driven userver

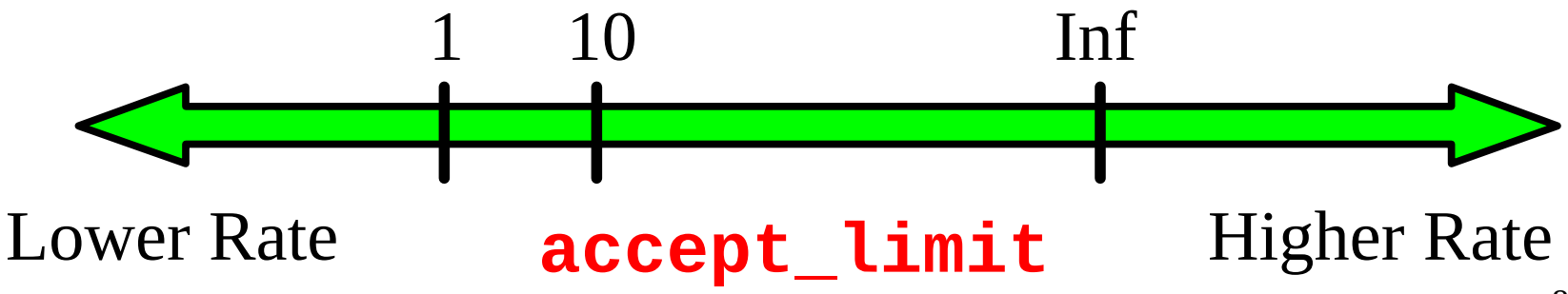


How to `accept()` new connections

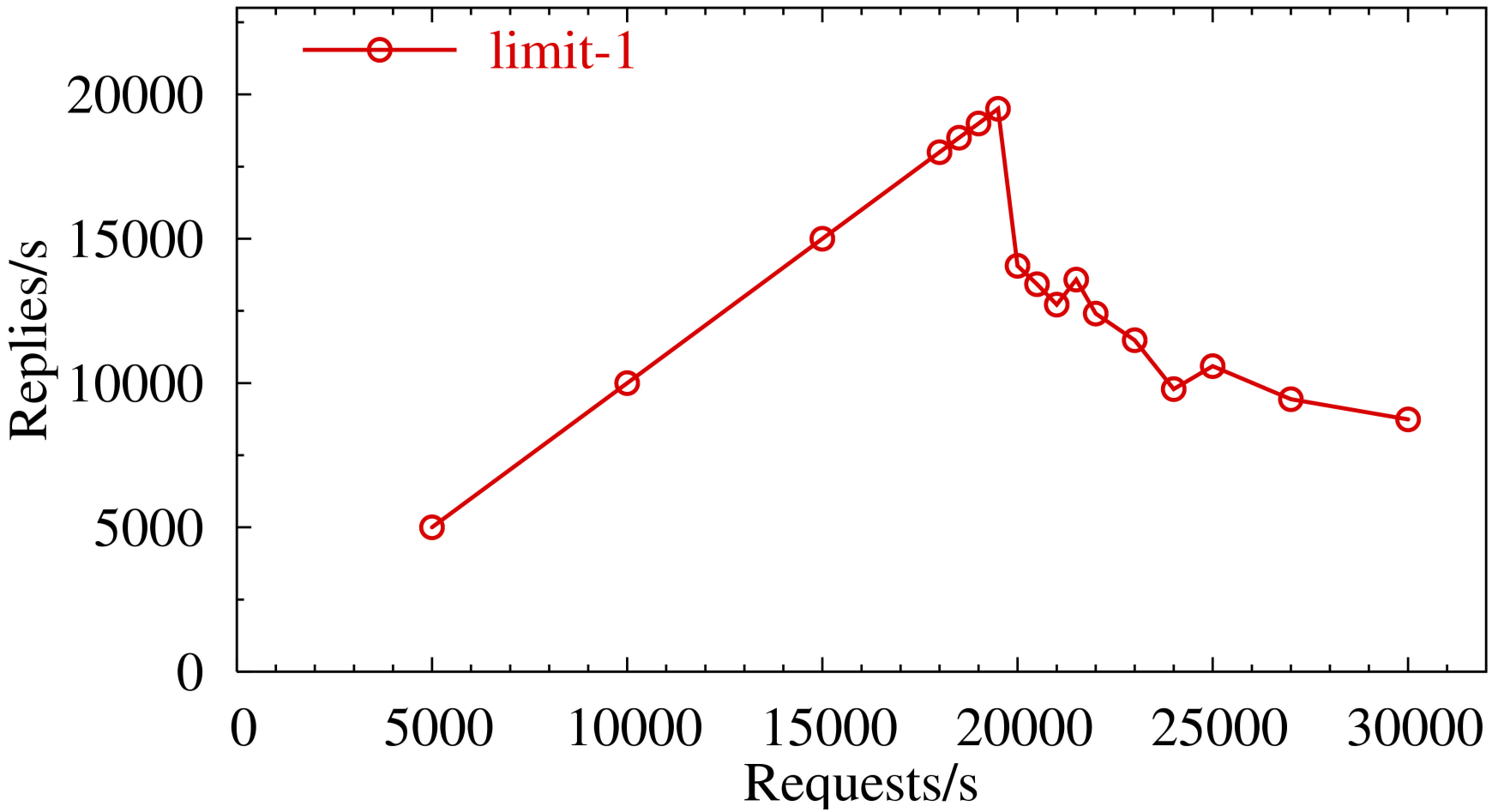
```
new_count = 0;
while (new_count < accept_limit) {
   new_sd = accept(sd, &addr, &len);
   if (new_sd < 0) {
      break;
   } else {
     setup_connection(new_sd);
     new_count++;
   }
}
return (new_count);
```

How to `accept()` new connections

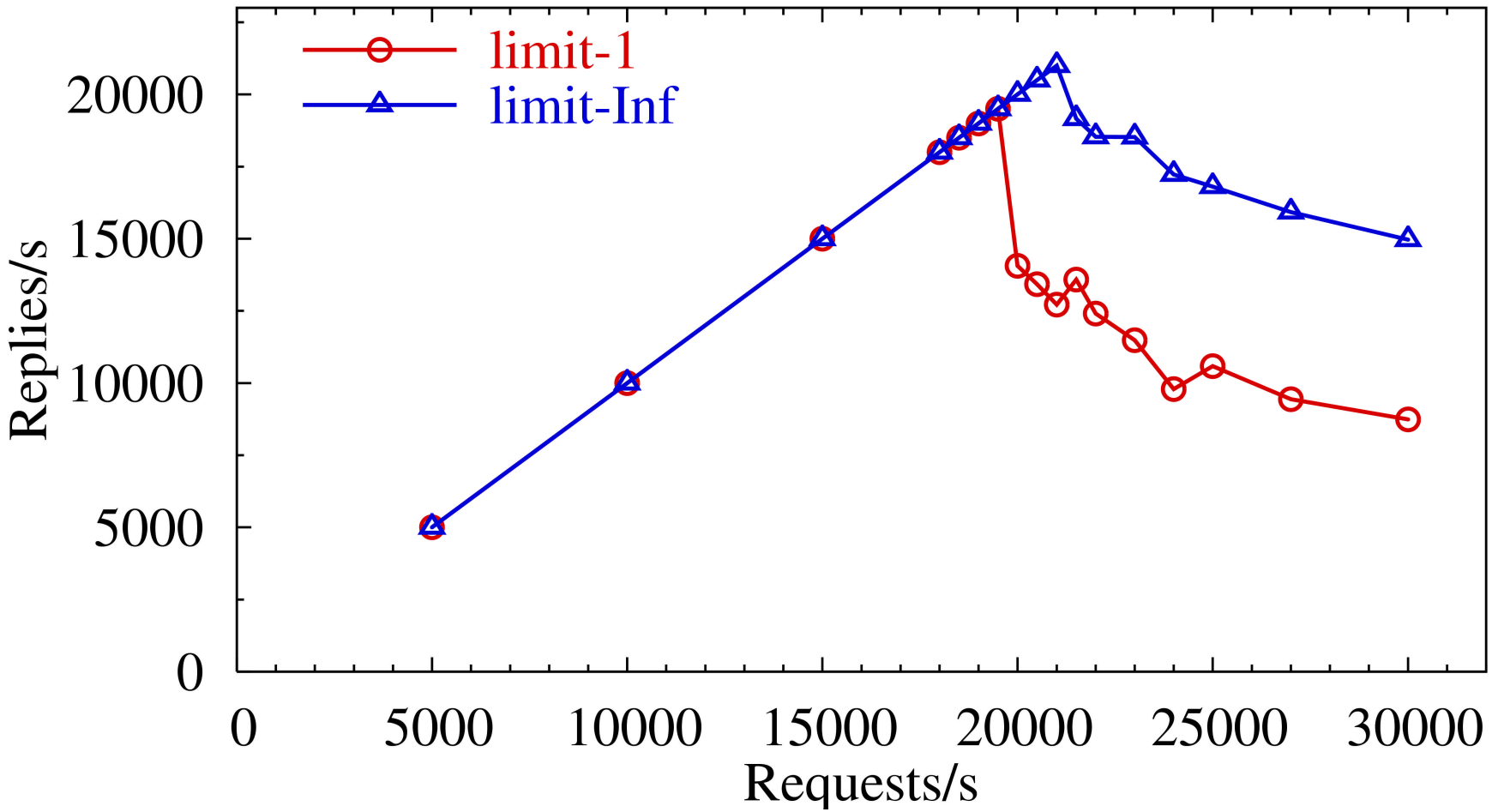
```
new_count = 0;
while (new_count < accept_limit) {
   new_sd = accept(sd, &addr, &len);
   if (new_sd < 0) {
      break;
   } else {
     setup_connection(new_sd);
     new_count++;
   }
}
return (new_count);
```



`accept()` strategies



`accept()` strategies



[Brecht et al., USENIX 2004]

select – based server

```
rdset = readfds; wrset = writefds;
n = select(max, &rdset, &wrset, ...);
for (i=0; i<max; i++) {
  sd = i;
  if (FD_ISSET(sd, &rdset))
    if (sd == listen_sd)
      x = accept_loop(listen_sd, limit);
    else
      x = do_read(sd, ...);
  if (FD_ISSET(sd, &wrset))
    x = do_sendfile(sd, ...);
}
```

Changing interest

```
FD_SET(sd, &readfds);
FD_CLR(sd, &writefds);
```

poll – based server

```
n = poll(parray, max, ...);
for (i=0; i<max; i++) {
  sd = parray[i].fd;
  if (parray[i].revents & POLLIN)
    if (sd == listen_sd)
      x = accept_loop(listen_sd, limit);
    else
      x = do_read(sd, ...);
  if (parray[i].revents & POLLOUT)
    x = do_sendfile(sd, ...);
}
```

Changing interest

```
parray[i].fd = sd;
parray[i].events |= POLLIN;
parray[i].events &= ~POLLOUT;
```

epoll

- **`epfd = epoll_create(max_sds)`**
  - initialize
- **`epoll_ctl(epfd,epoll_op,sd,&evt)`**
  - add/delete/change interest in events on sd
  - **`EPOLL_CTL_ADD`**
  - **`EPOLL_CTL_MOD`**
  - **`EPOLL_CTL_DEL`**
- **`epoll_wait(epfd,results,max,timeo)`**
  - get events of interest

epoll

- **`epfd = epoll_create(max_sds)`**
  - initialize
- **`epoll_ctl(epfd,epoll_op,sd,&evt)`**
  - add/delete/change interest in events on sd
  - **`EPOLL_CTL_ADD`**
  - **`EPOLL_CTL_MOD`**
  - **`EPOLL_CTL_DEL`**
- **`epoll_wait(epfd,results,max,timeo)`**
  - get events of interest

Separates declaration of interest from getting events

epoll – based server

```
n = epoll_wait(epfd, results, max, ...);
for (i=0; i<n; i++) {
  sd = results[i].data.fd
  if (results[i].events & POLLIN)
    if (sd == listen_sd)
      x = accept_loop(listen_sd, limit);
    else
      x = do_read(sd, ...);
  if (results[i].events & POLLOUT)
    x = do_sendfile(sd, ...);
}
```

Changing interest

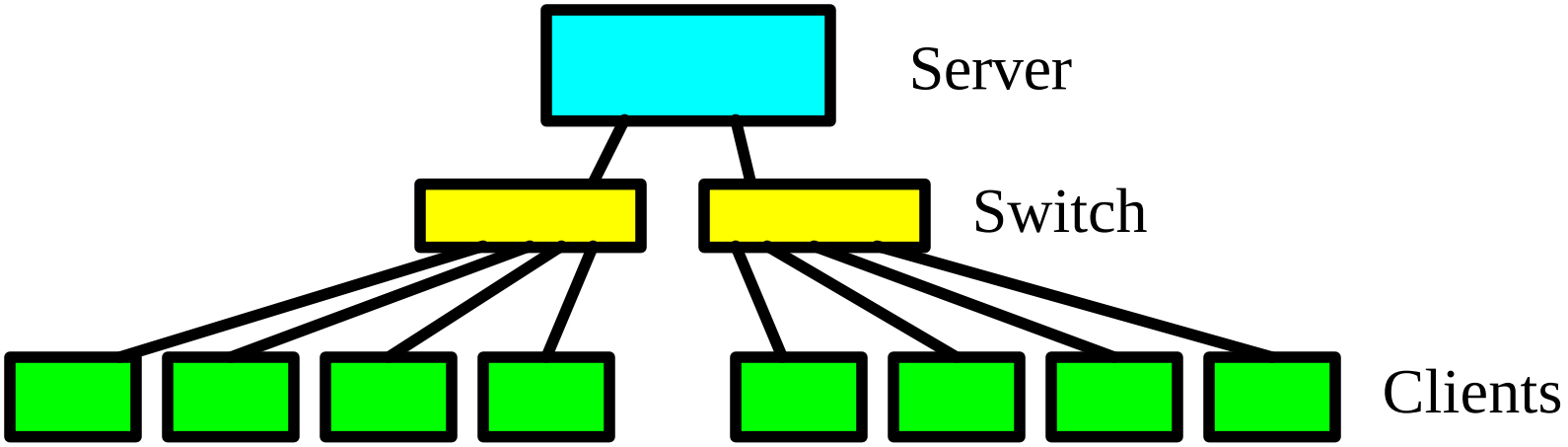
```
evt.data.fd = sd; evt.events = POLLIN;
epoll_ctl(epfd, EPOLL_CLT_MOD, sd, &evt);
epoll_ctl(epfd, EPOLL_CTL_DEL, sd, &evt);
```

Level-Triggered and Edge-Triggered epoll

- **Level-Triggered**
  - provides current state of the sd
    - e.g., readable, writable
  - consecutive calls: same result for sds w/o changes
  - select, poll, epoll level-triggered work this way
- **Edge-triggered**
  - provides changes in state of sd since last call
    - e.g., transitioned to readable / writable
  - consecutive calls: no results for sds w/o changes
  - application needs to track current state

Environment

- **Server**
  - `userver`
  - Linux 2.6.5
- **Hardware**
  - 2.4 GHz Xeon, 1 GB RAM, 2 x 1 Gbps Enet
  - 4 clients connected to each 1 Gbps subnet



Workloads

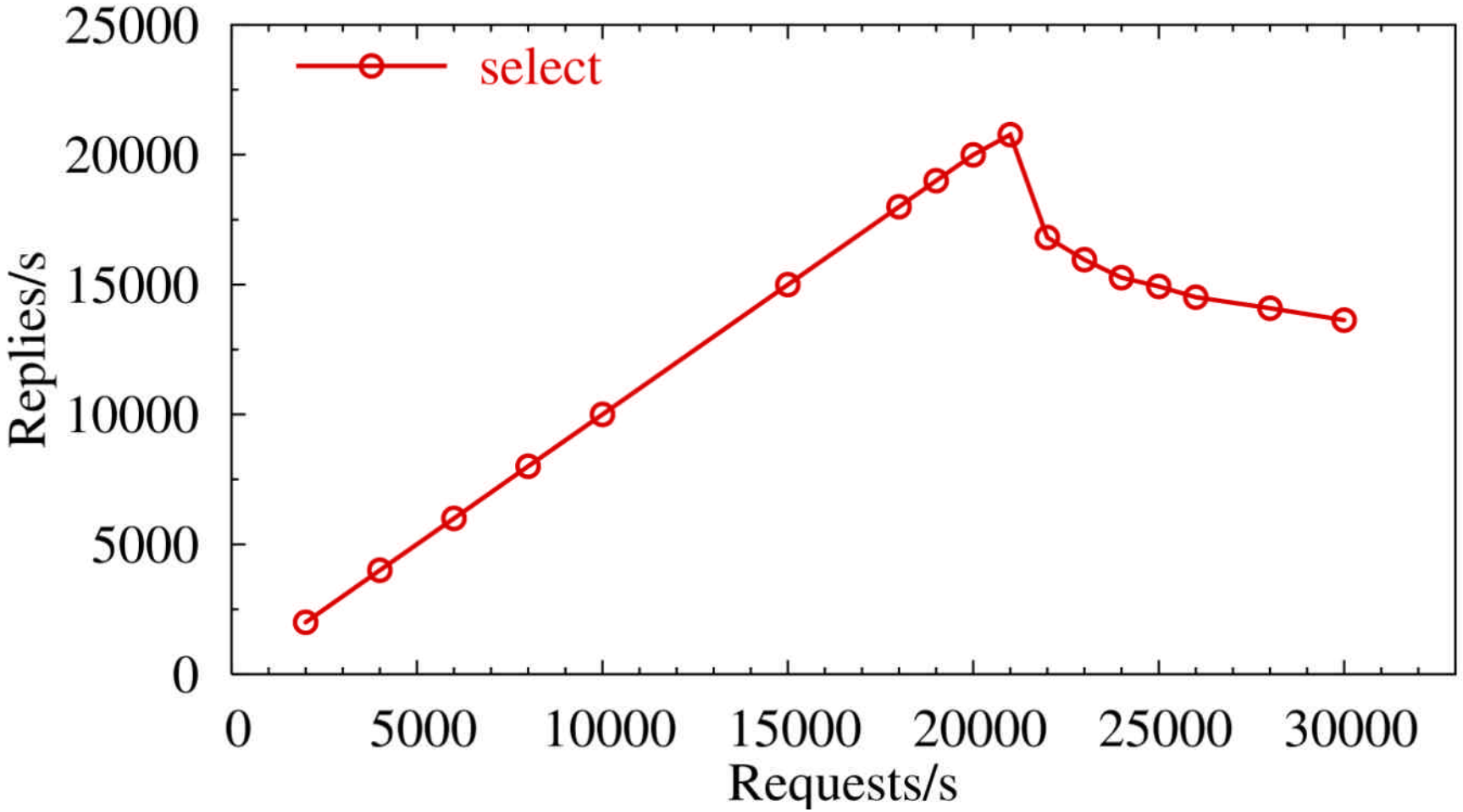
- **One byte workload**
  - all clients request single 1 byte file
  - hammers on / stresses the event mechanism
- **Static SPECWeb99-like**
  - clients request multiple files
  - sizes and distributions from specweb99 benchmark
  - fits in memory (avoid disk activity, more stress)
- **`httperf`**
  - workload generator generates overload conditions

Workloads

- **One byte workload**
  - all clients request single 1 byte file
  - hammers on / stresses the event mechanism
- **Static SPECWeb99-like**
  - clients request multiple files
  - sizes and distributions from specweb99 benchmark
  - fits in memory (avoid disk activity, more stress)
- **`httperf`**
  - workload generator generates overload conditions

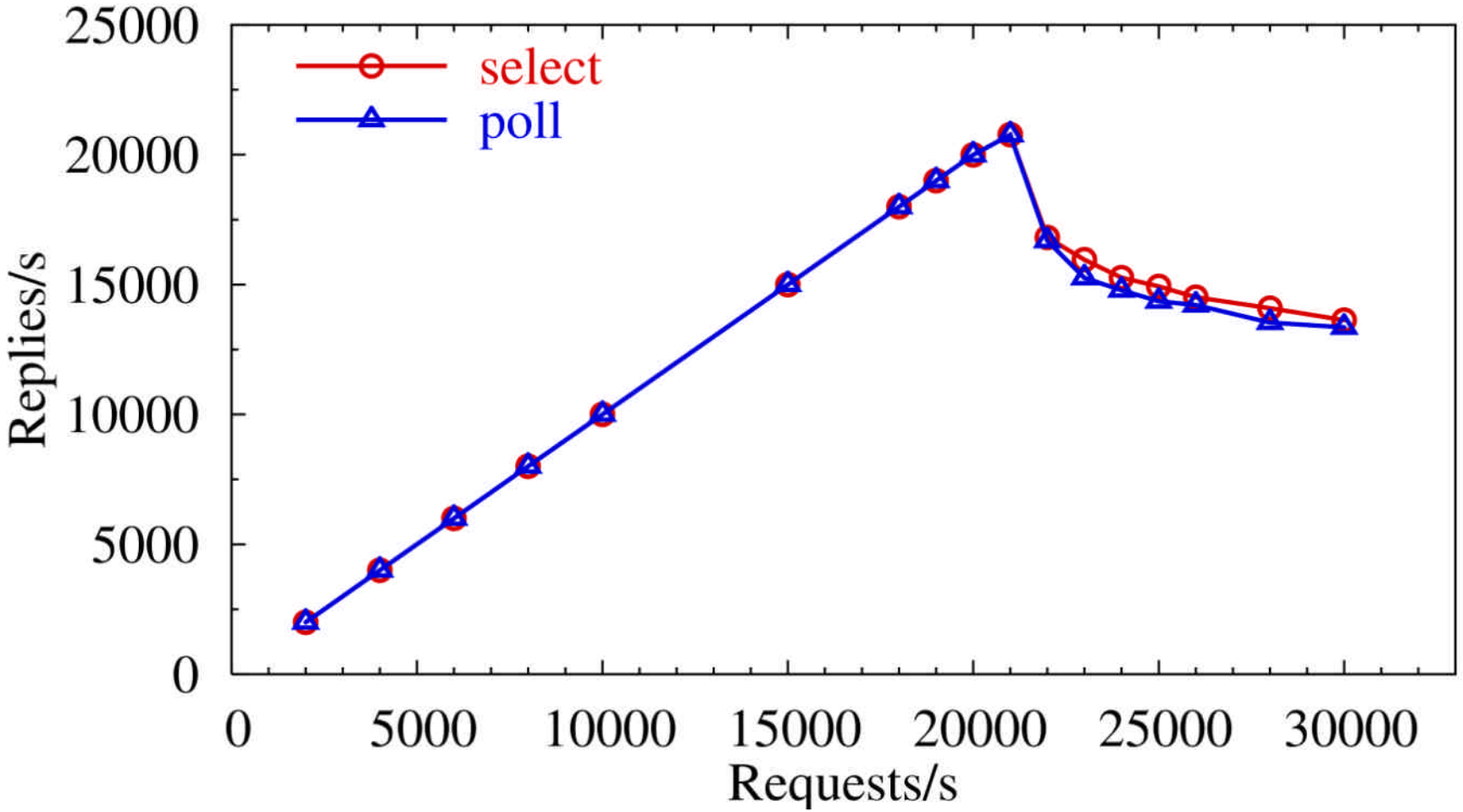
Experiments with and without idle connections

Performance Comparison



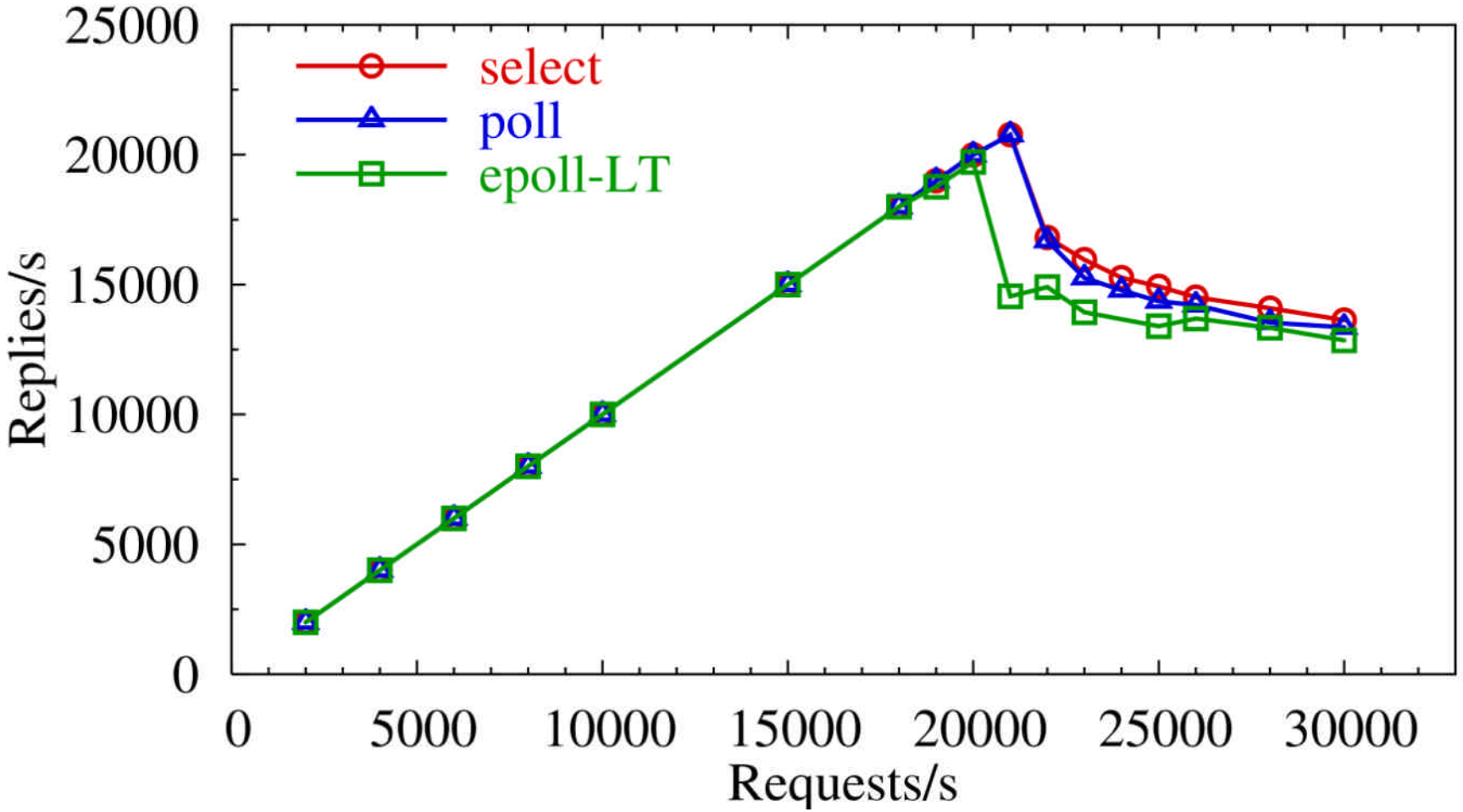
- 1 byte file workload without idle connections

Performance Comparison



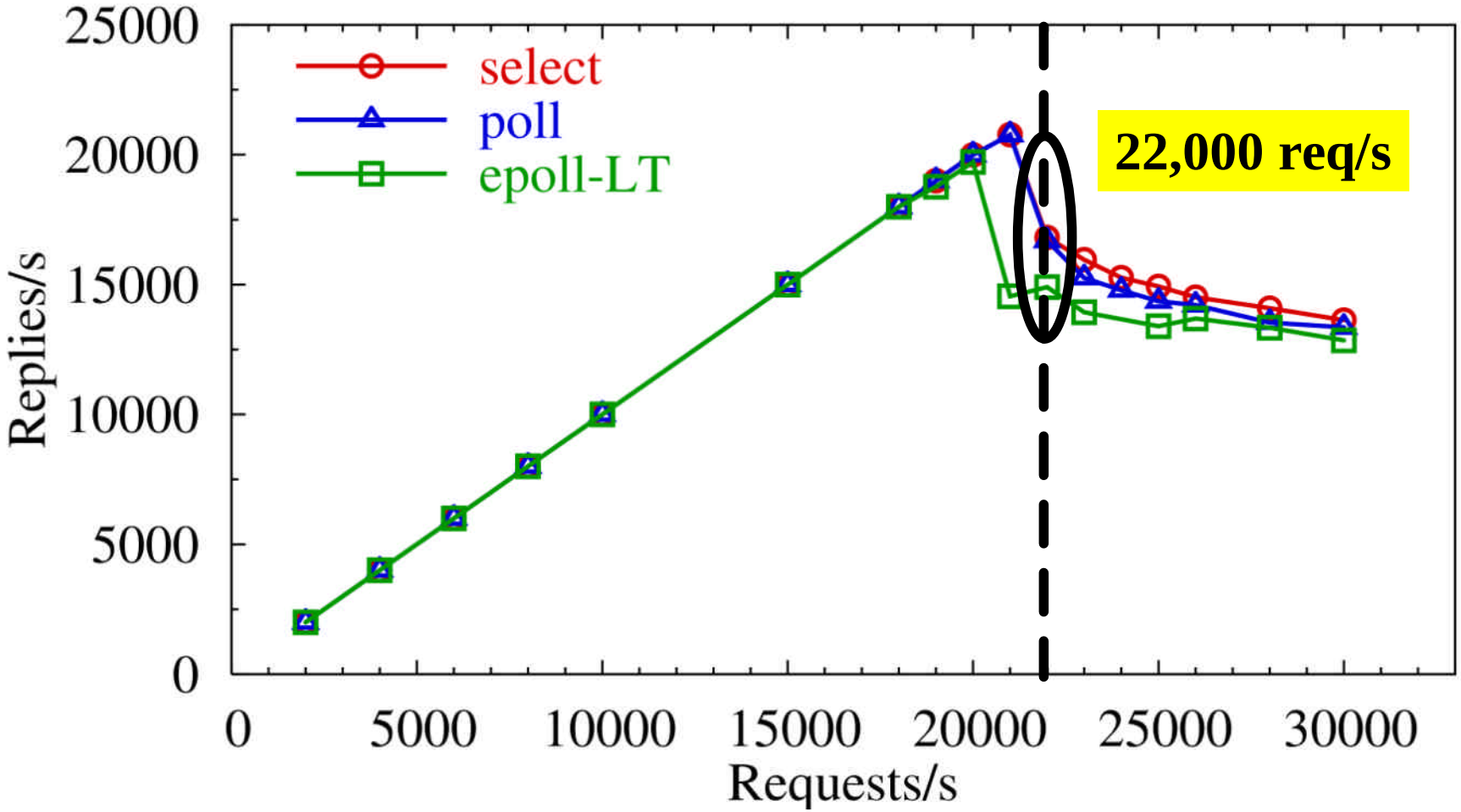
- 1 byte file workload without idle connections

Performance Comparison



- 1 byte file workload without idle connections

Performance Comparison



- 1 byte file workload without idle connections

gprof comparison

Syscall	select	epoll-LT	poll
read	21.51	20.95	20.97
close	14.90	14.05	14.79
select	13.33		
poll			13.32
epoll_wait		7.15	
epoll_ctl		16.34	
setsockopt	11.17	9.13	10.68
accept	10.08	9.51	10.20
write	5.98	5.06	5.70
fcntl	3.66	3.34	3.61
sendfile	3.43	2.70	3.43

Reducing `epoll_ctl()` overhead

- **Don't modify interests** (`epoll2`)
  - when accepting connection add to interest set
    - interest in reading and writing (w/o blocking)
  - when closing connection remove from interest set
  - less `epoll_ctl` but more `epoll_wait`
  - `epoll_wait` returns when >= one evt of interest

Reducing `epoll_ctl()` overhead

- **Aggregate System Calls** **`(epoll_ctlv)`**
  - system call -- uses an array of sds/interest sets
  - one system call, many changes (ala readv/writev)
  - essentially the same work but fewer calls
  - deleting from interest set at close
    - use **`epoll_ctlv`** (requires delaying close)
    - just let **`close`** remove sd from interest set
    - call **`epoll_ctl`** explicitly

Reducing `epoll_ctl()` overhead

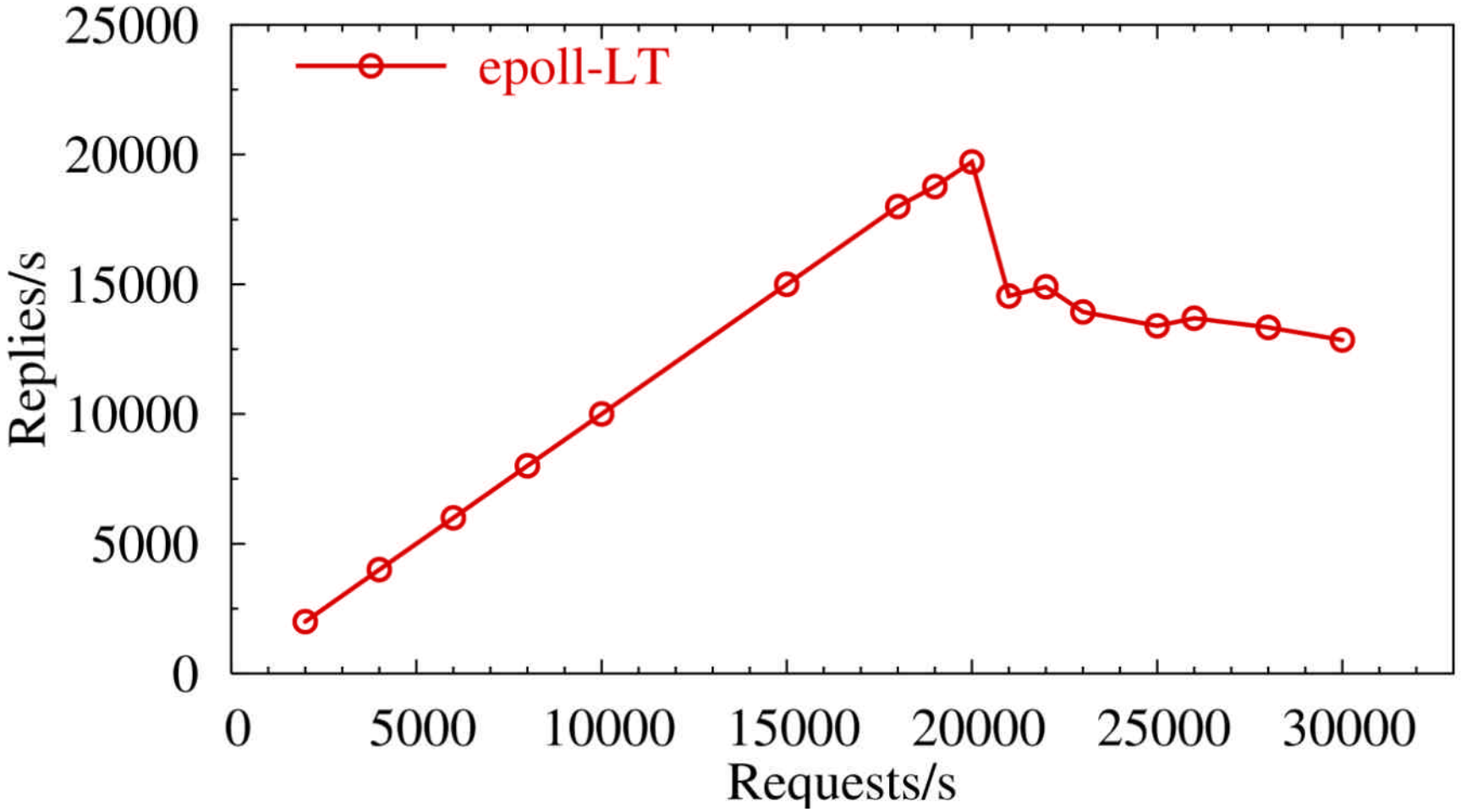
- **Aggregate System Calls** **`(epoll_ctlv)`**
  - system call -- uses an array of sds/interest sets
  - one system call, many changes (ala readv/writev)
  - essentially the same work but fewer calls
  - deleting from interest set at close
    - use **`epoll_ctlv`** (requires delaying close)
    - just let **`close`** remove sd from interest set
    - call **`epoll_ctl`** explicitly

    - No performance difference between last two

Reducing `epoll_ctl()` overhead

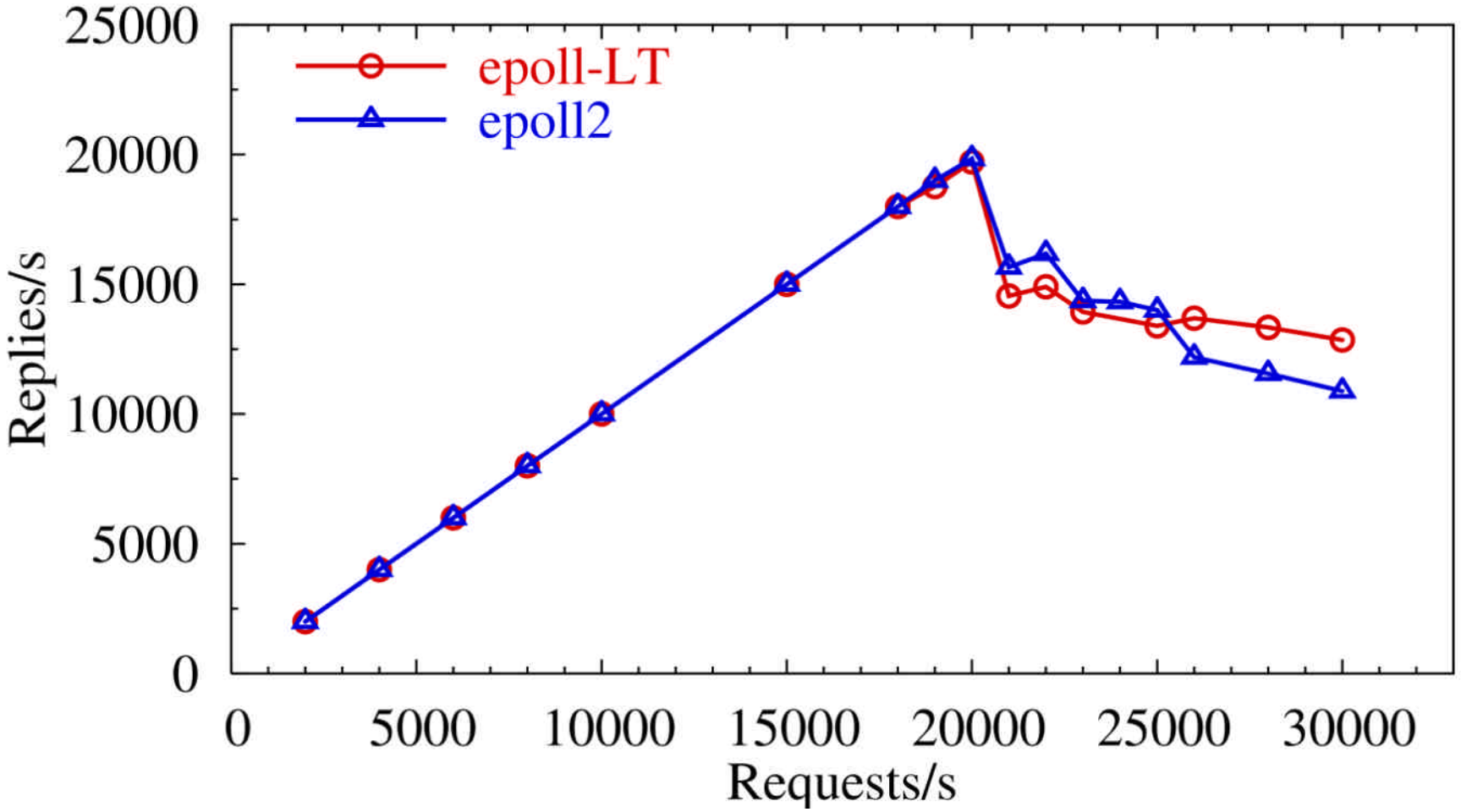
- **Edge-triggered** **`(epoll-ET)`**
  - when accepting connection add to interest set interest in reading and writing (w/o blocking)
  - when closing connection remove from interest set
  - **`epoll_wait`** returns events on change on sd
  - requires tracking state of the sd in application

Reducing `epoll_ctl` calls



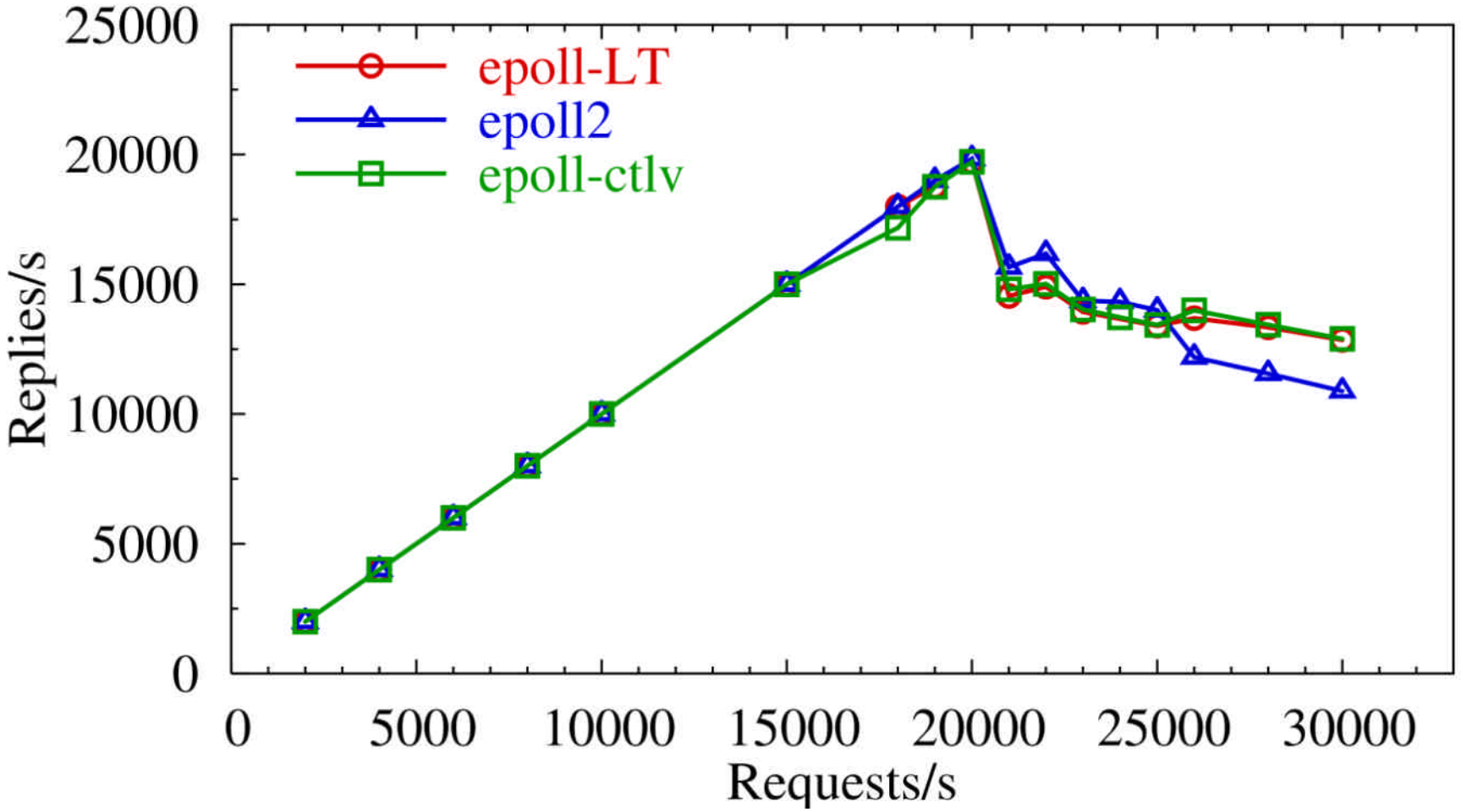
- 1 byte file workload without idle connections

Reducing `epoll_ctl` calls



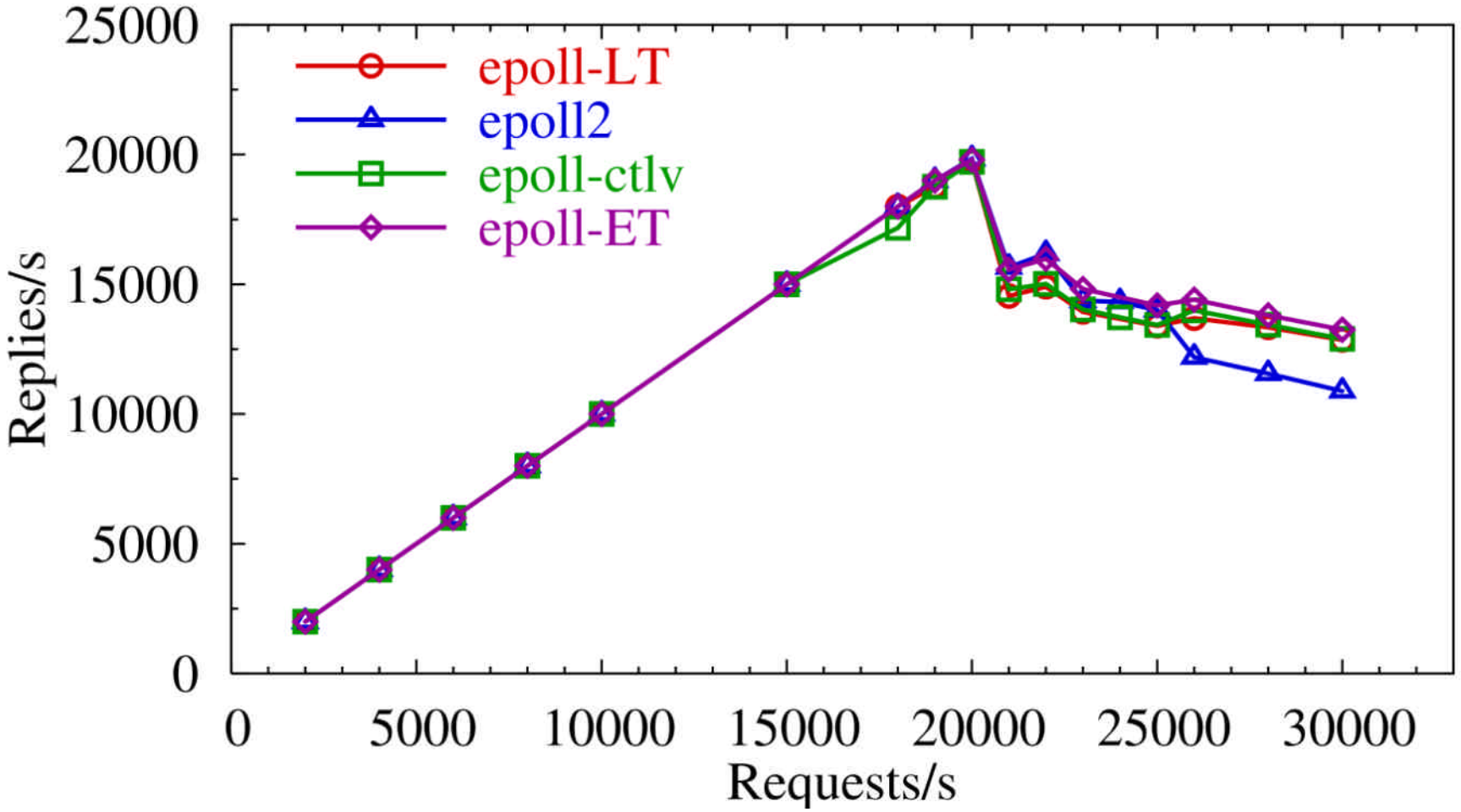
- 1 byte file workload without idle connections

Reducing `epoll_ctl` calls



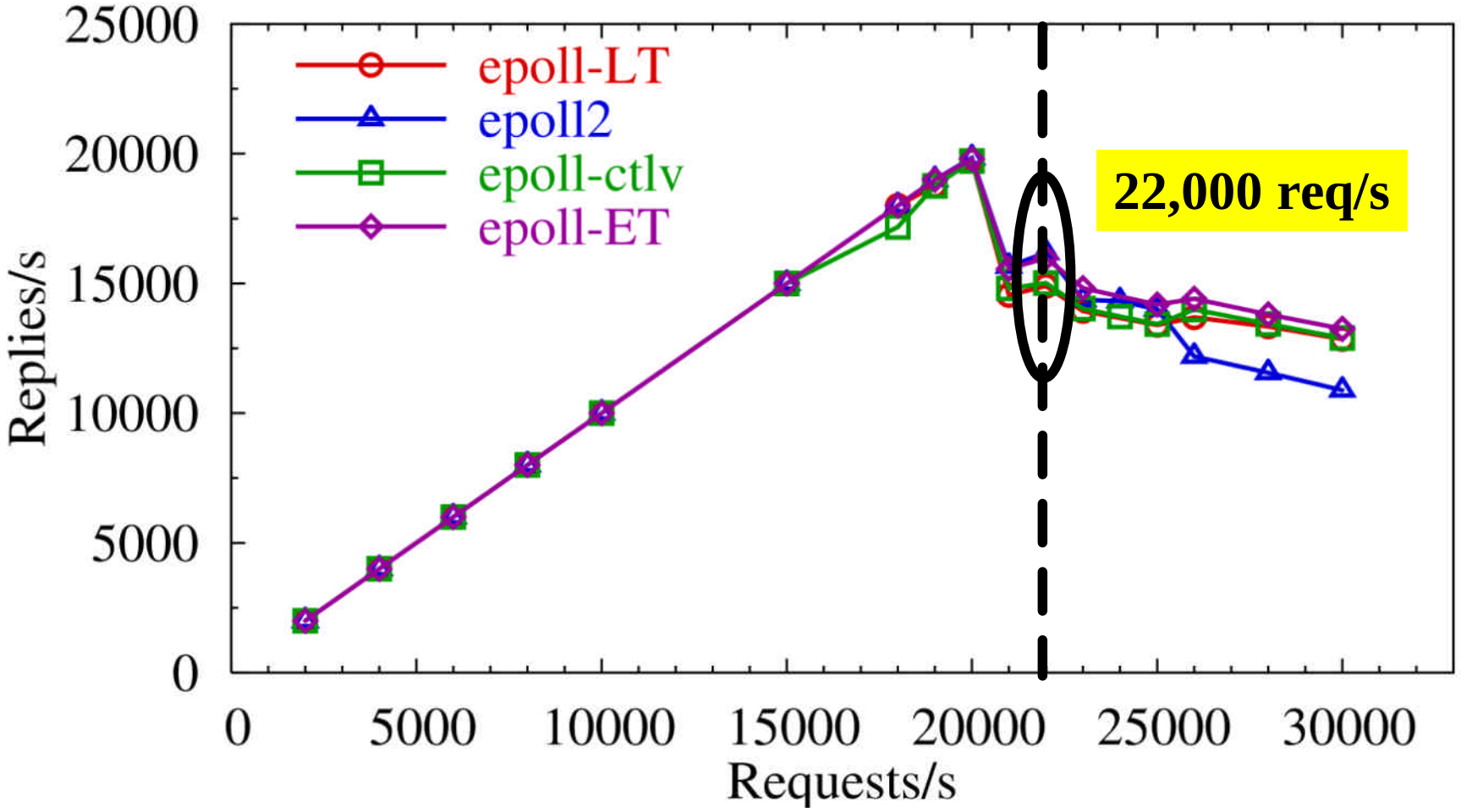
- 1 byte file workload without idle connections

Reducing `epoll_ctl` calls



- 1 byte file workload without idle connections

Reducing `epoll_ctl` calls



- 1 byte file workload without idle connections

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

gprof results @ 22,000 req/s

Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

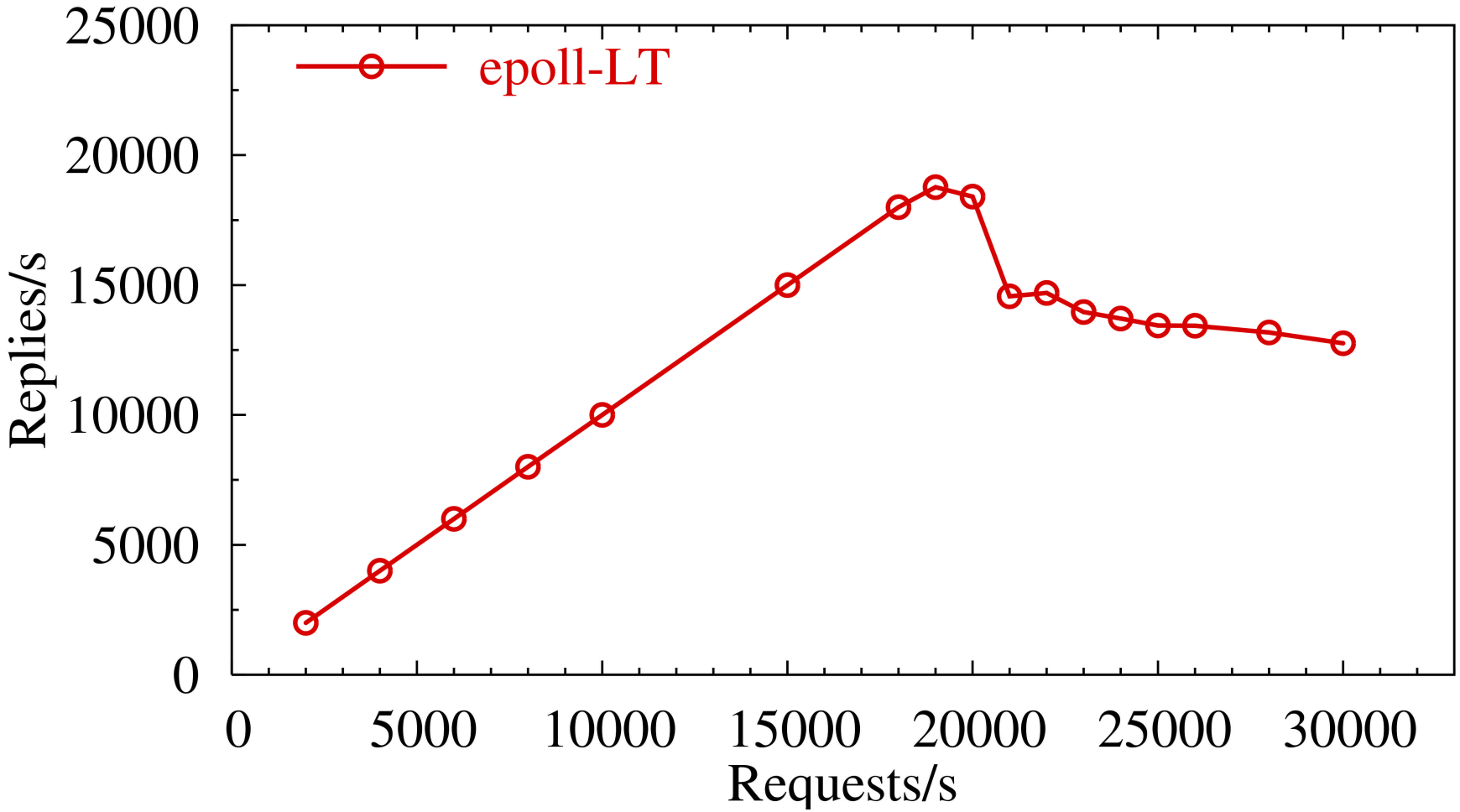
Values shown are percentage of time spent

gprof results @ 22,000 req/s

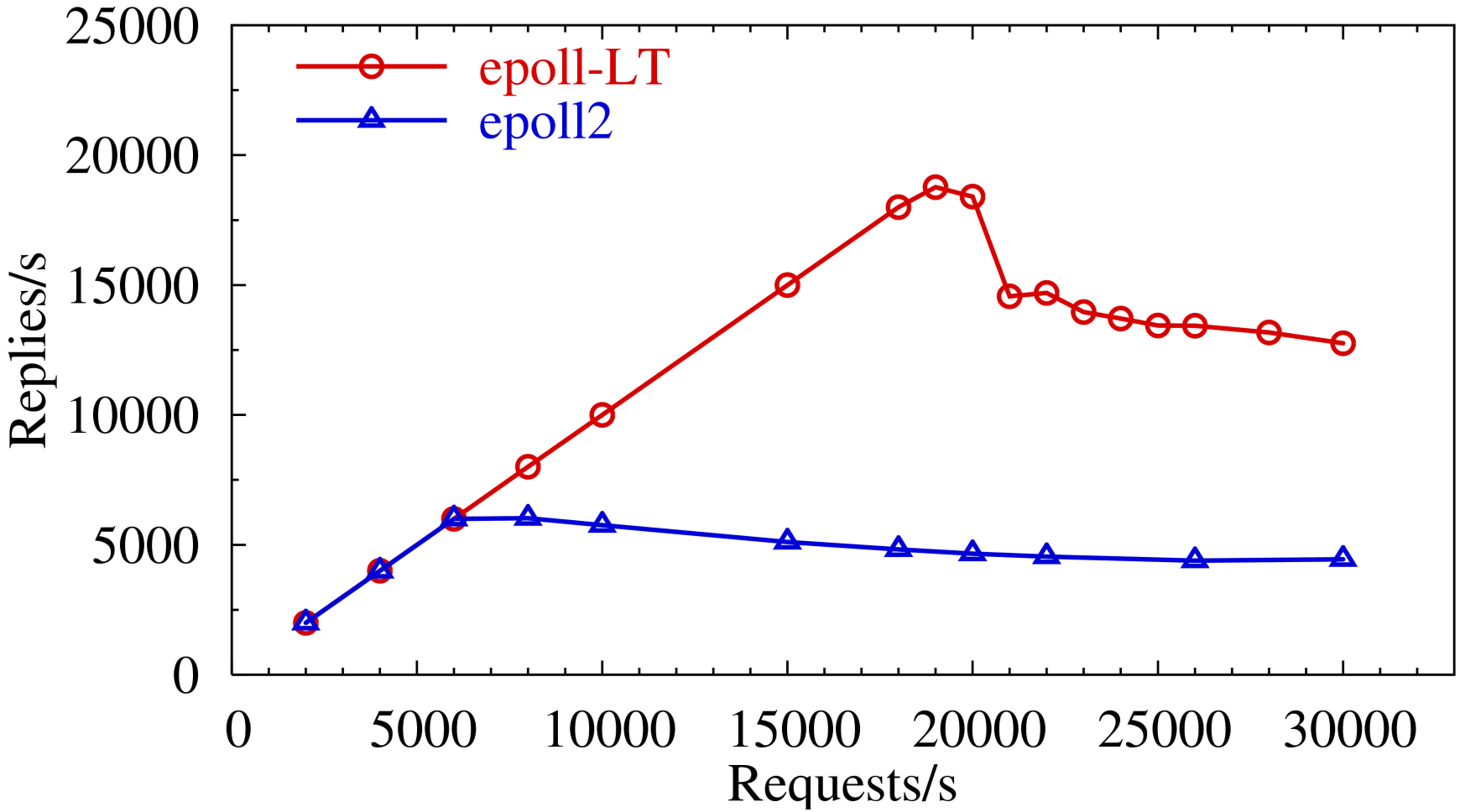
Syscall	epoll	epoll2	ctlv	edge
read	20.95	20.08	21.41	22.19
close	14.05	13.02	14.90	14.14
epoll_wait	7.15	12.56	6.01	6.52
epoll_ctl	16.34	10.27	5.98	11.06
epoll_ctlv			9.28	
sockopt	9.13	7.57	9.13	9.08
accept	9.51	9.05	9.76	9.30
write	5.06	4.13	5.10	5.31
fcntl	3.34	3.14	3.37	3.34
sendfile	2.70	3.00	2.71	3.91

Values shown are percentage of time spent

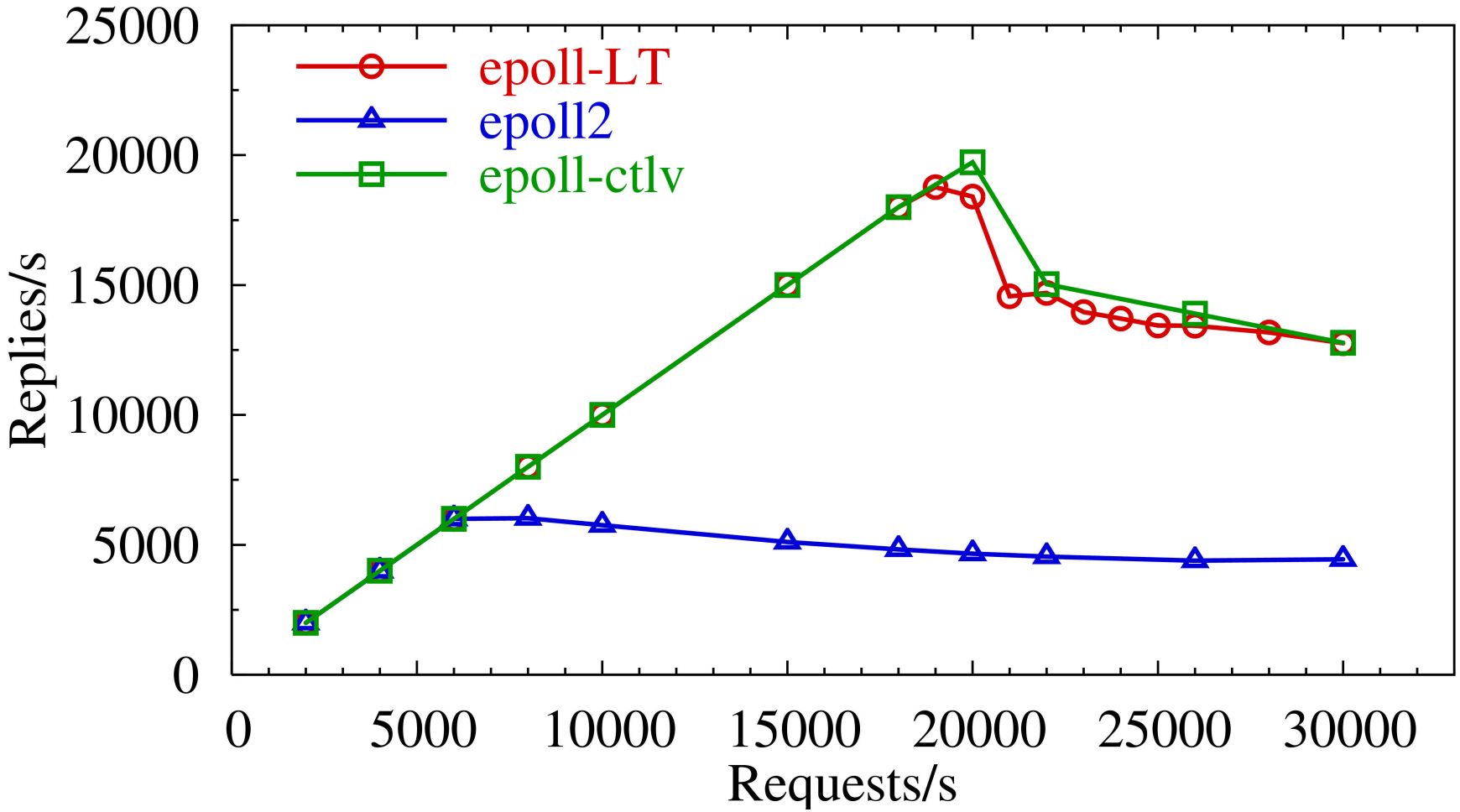
1 Byte File: 10,000 idle conns



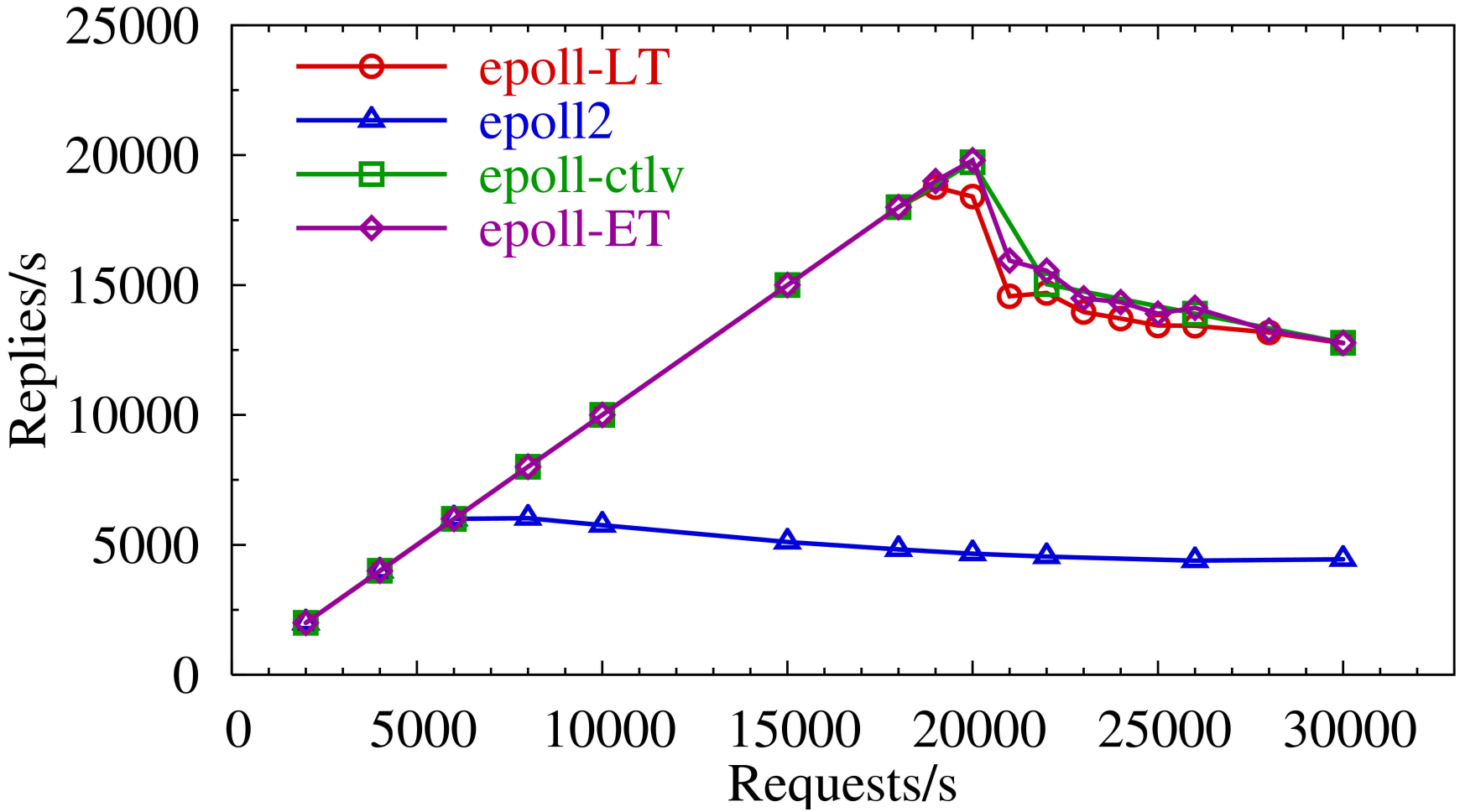
1 Byte File: 10,000 idle conns



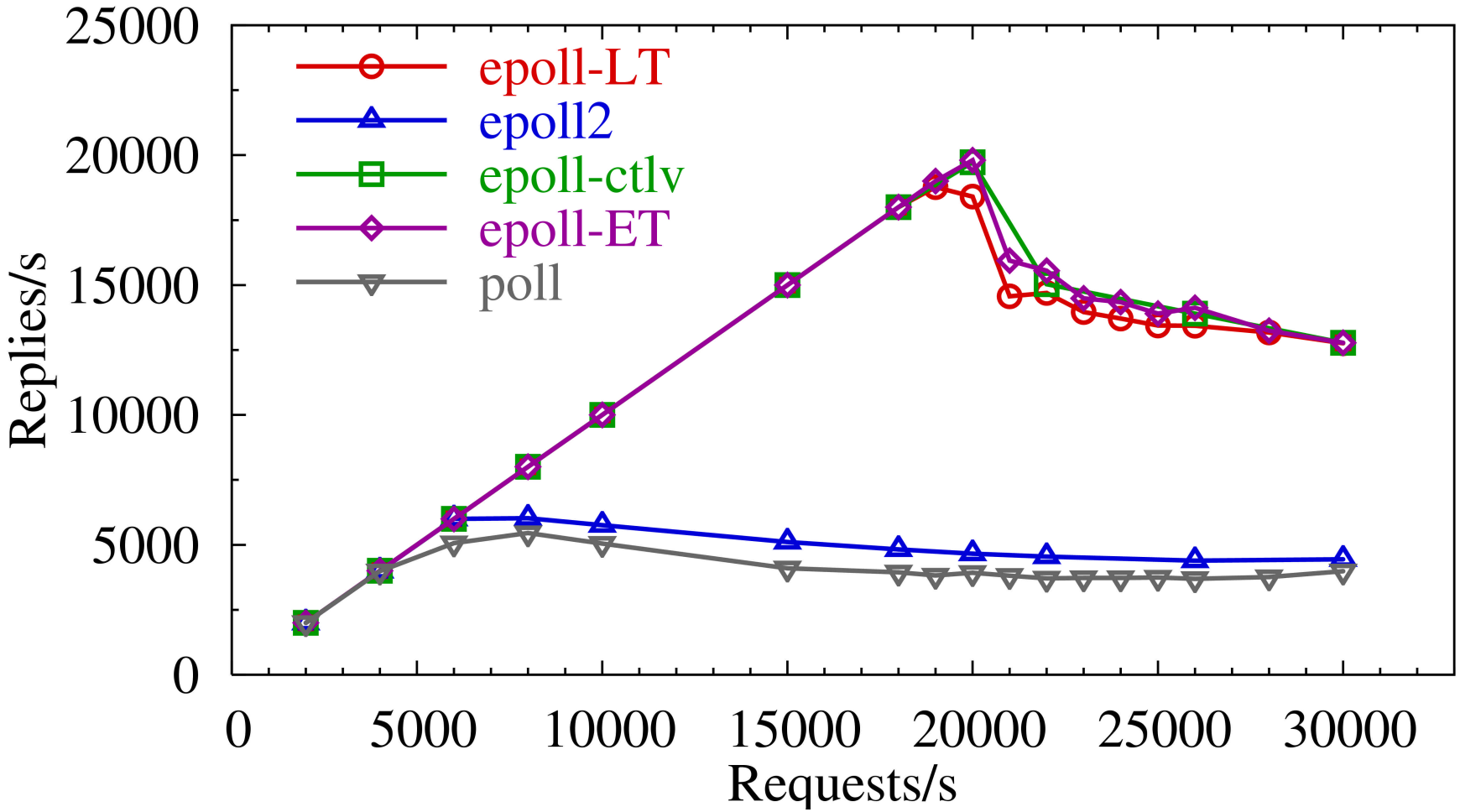
1 Byte File: 10,000 idle conns



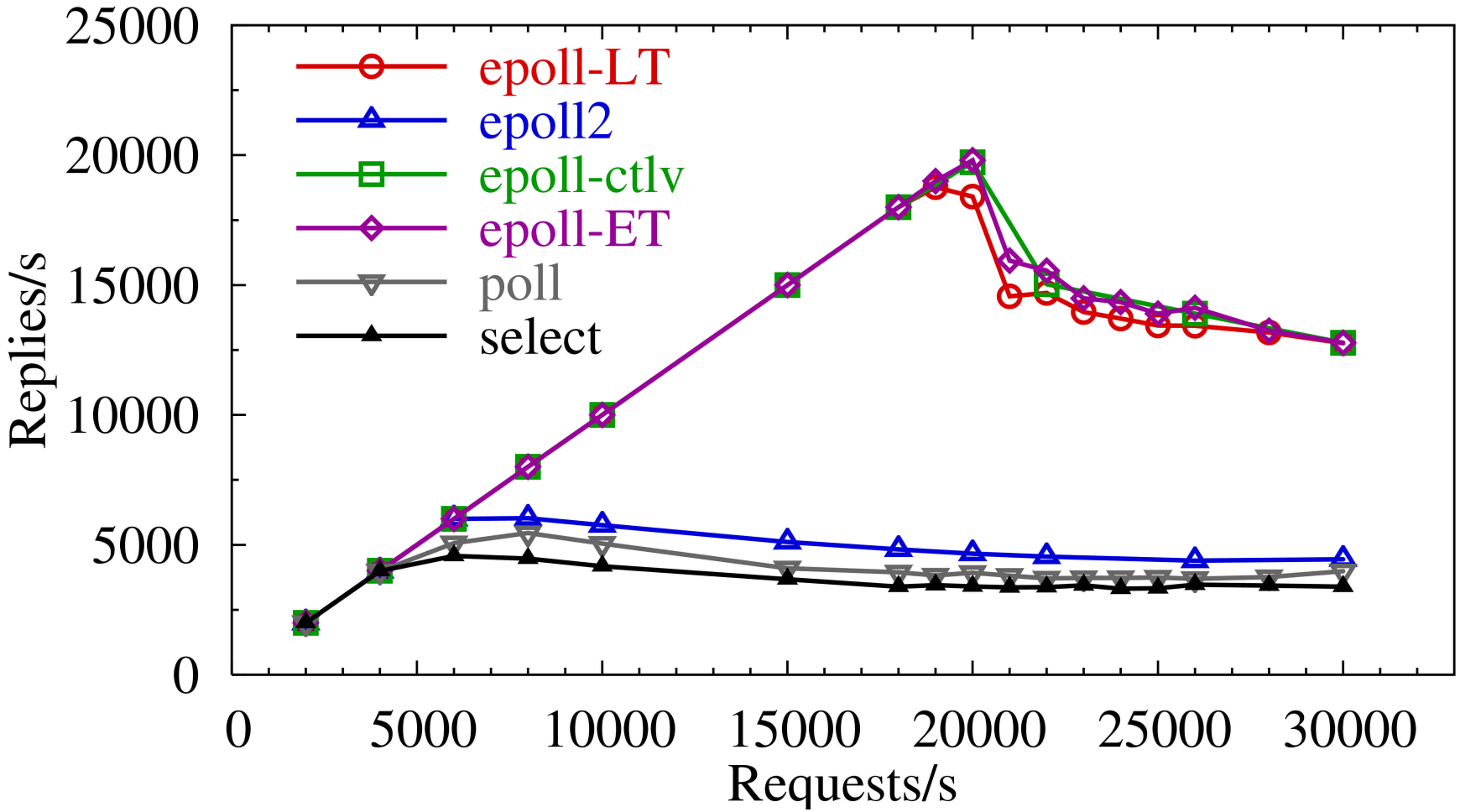
1 Byte File: 10,000 idle conns



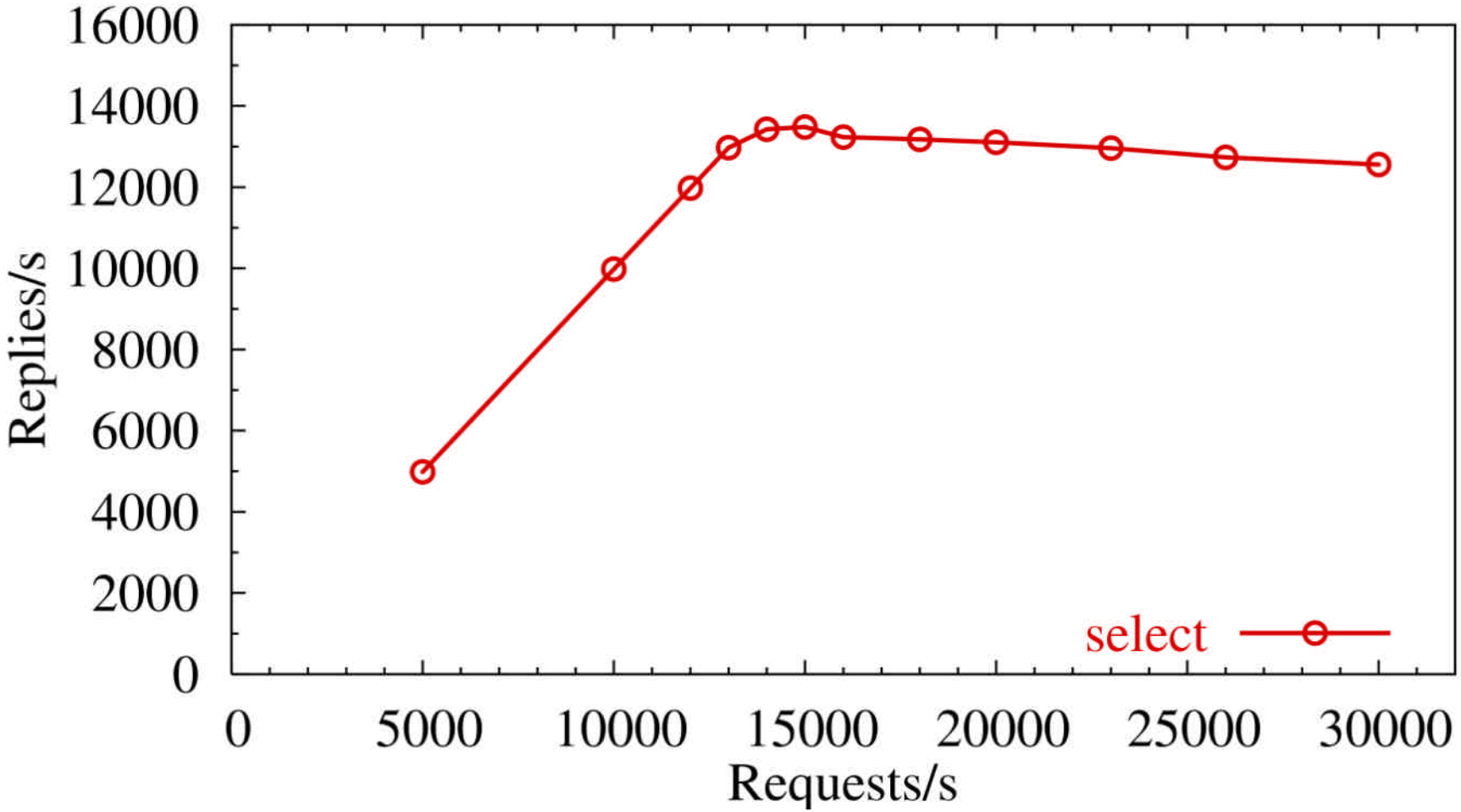
1 Byte File: 10,000 idle conns



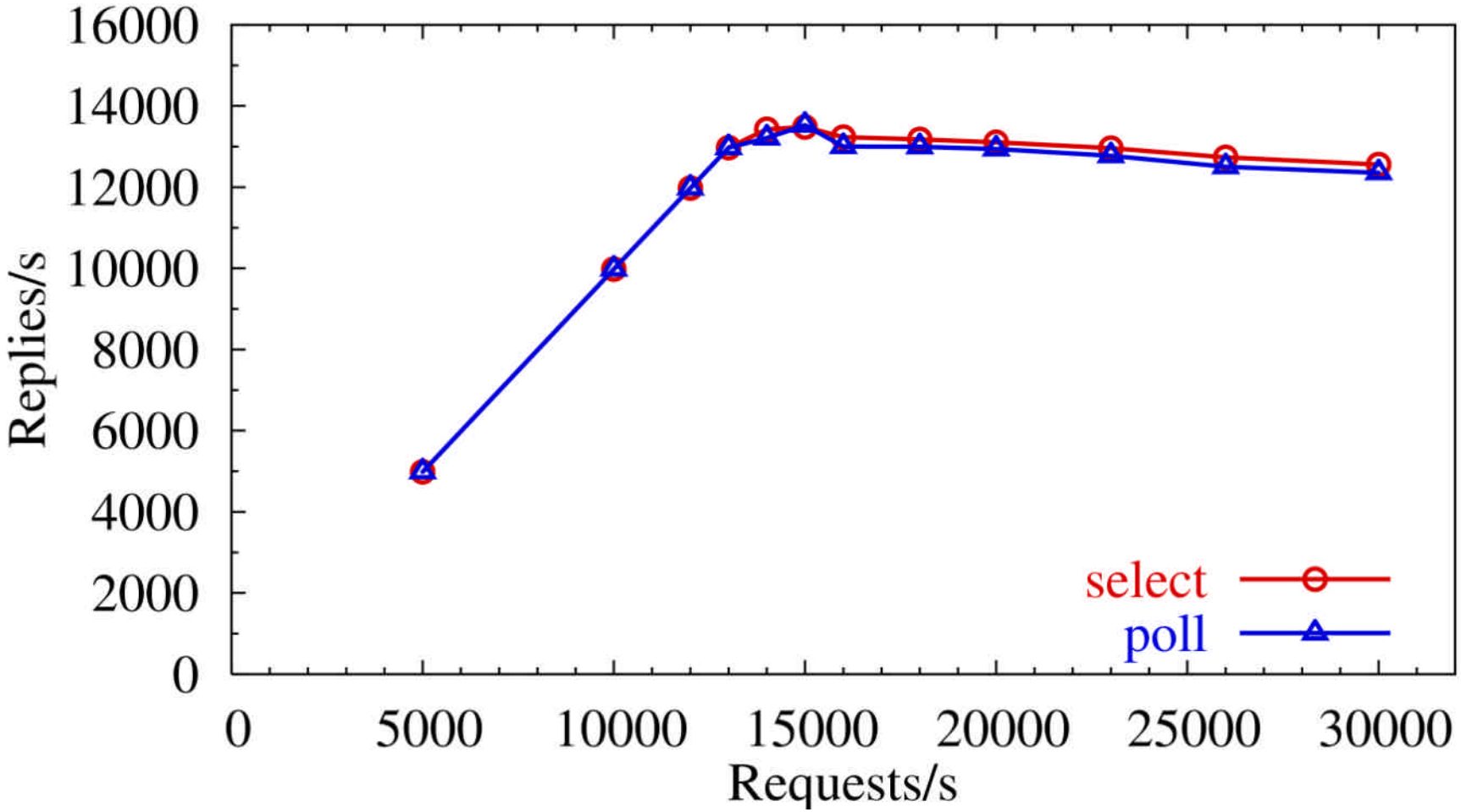
1 Byte File: 10,000 idle conns



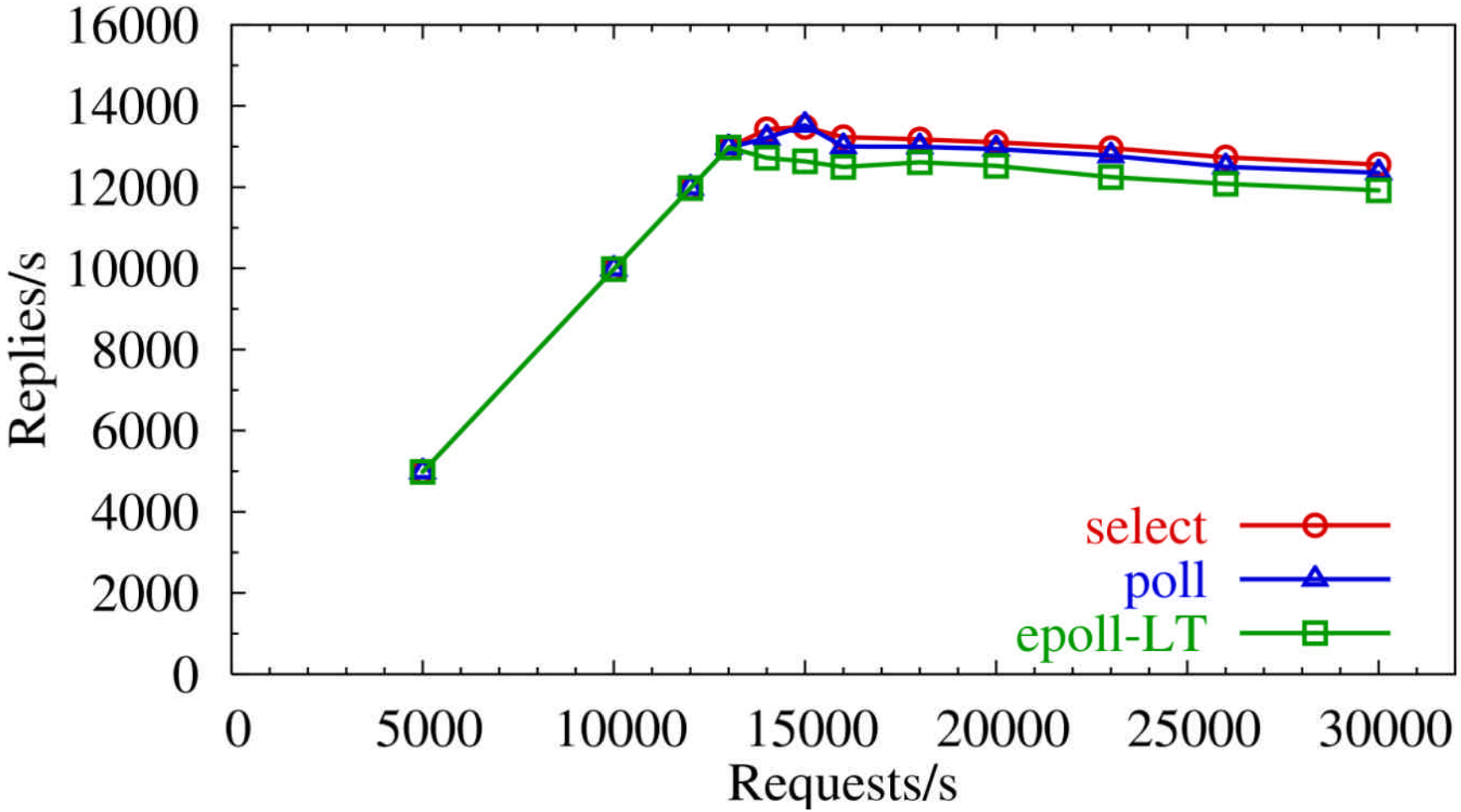
SPECweb99-like Workload w/o idle conns



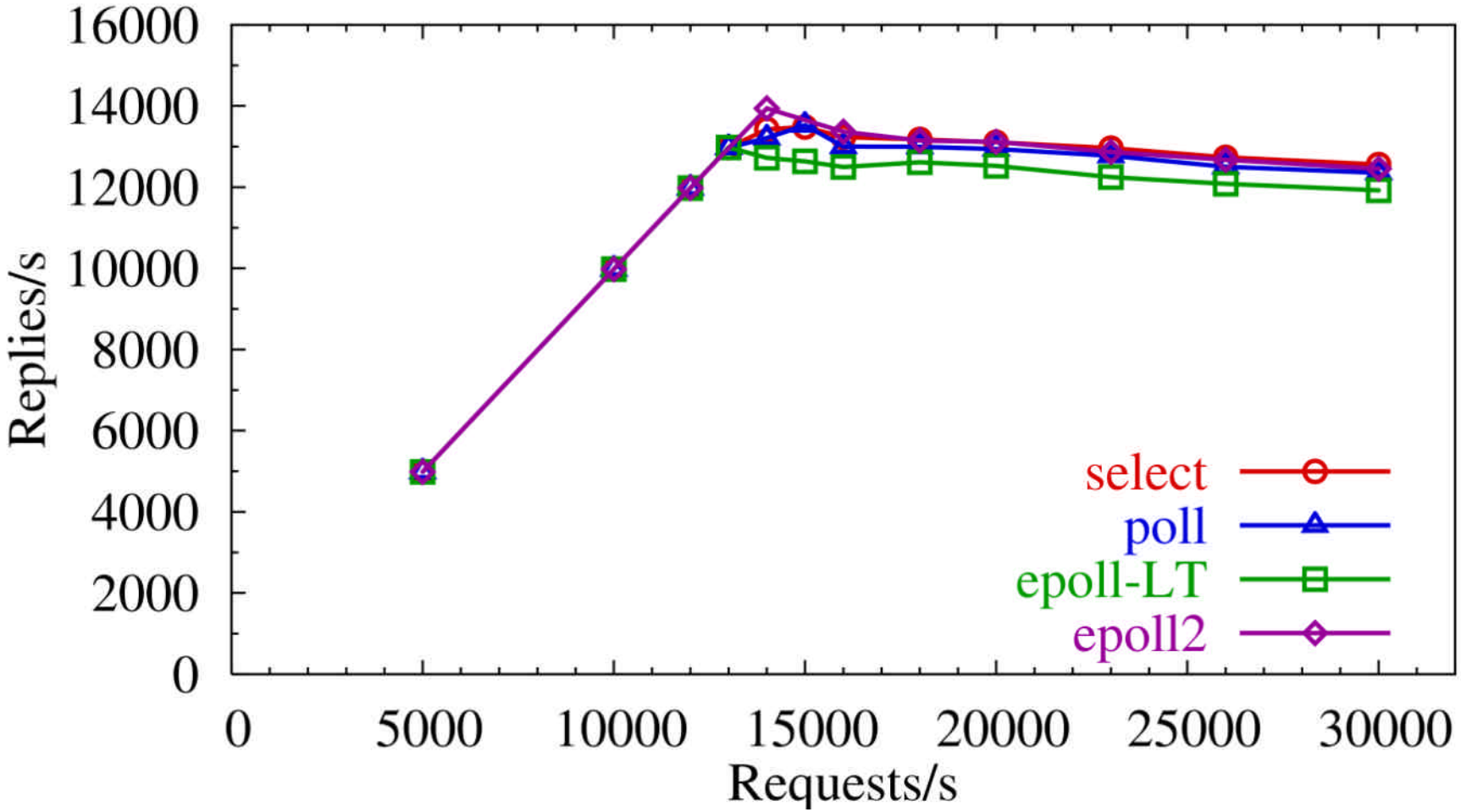
SPECweb99-like Workload w/o idle conns



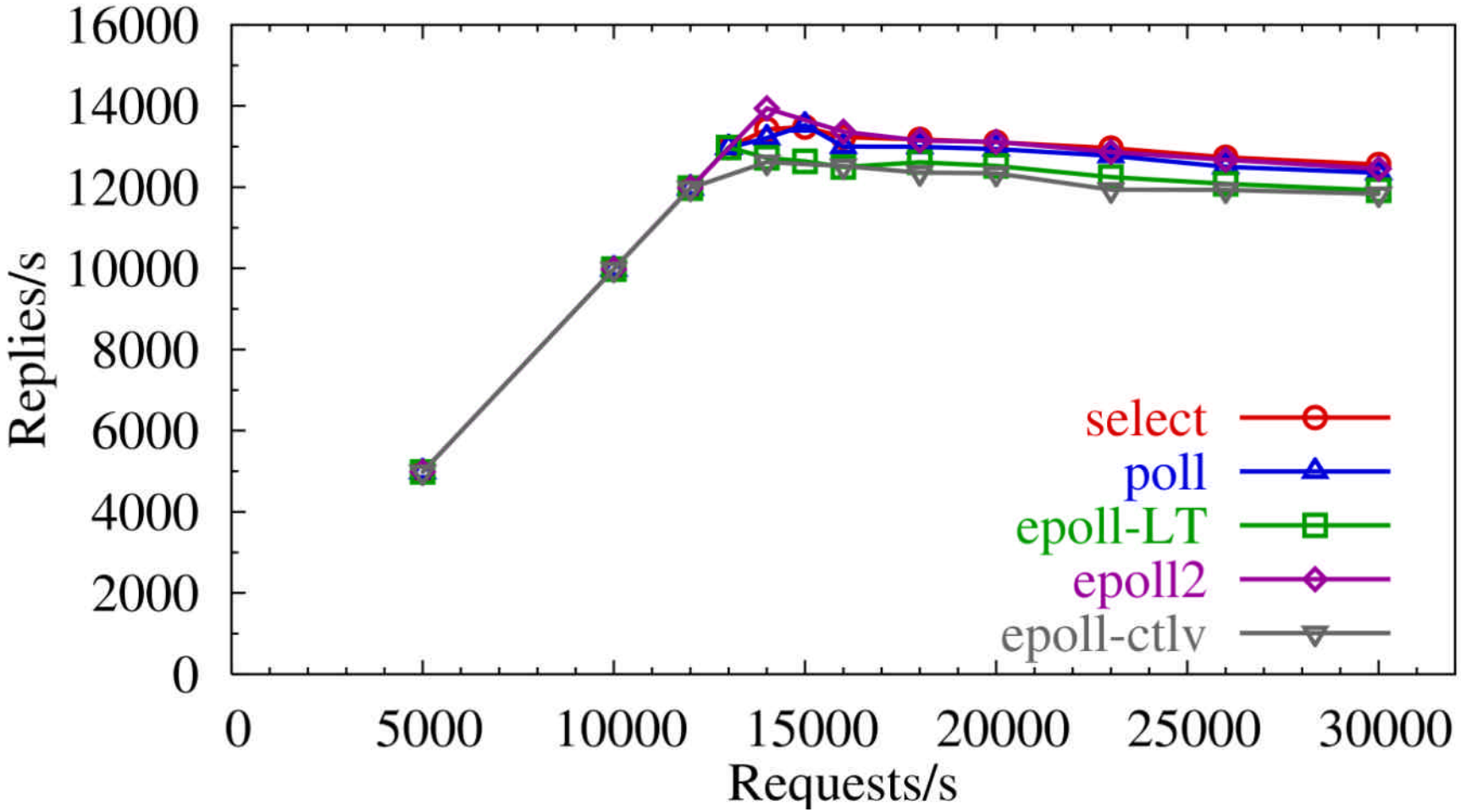
SPECweb99-like Workload w/o idle conns



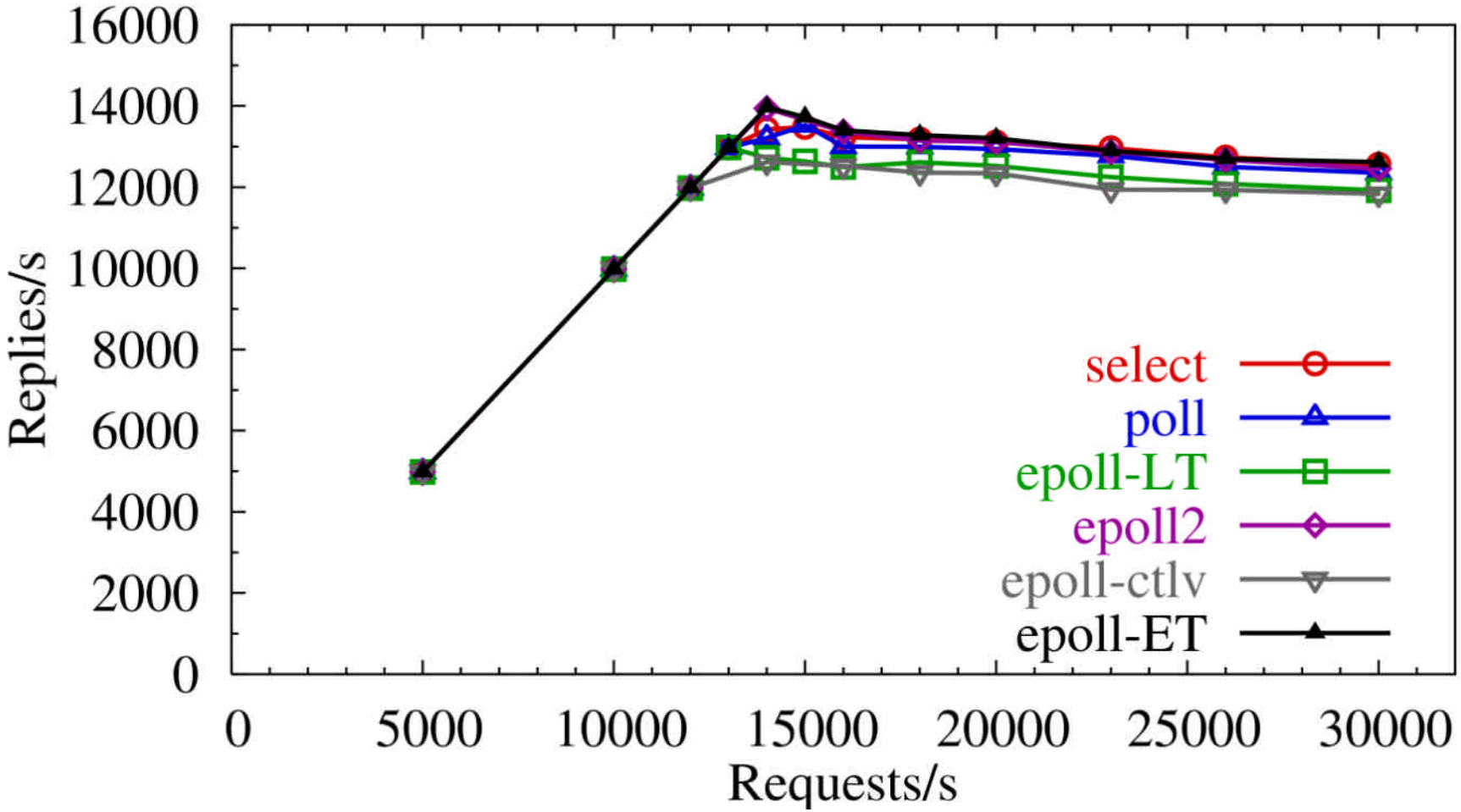
SPECweb99-like Workload w/o idle conns



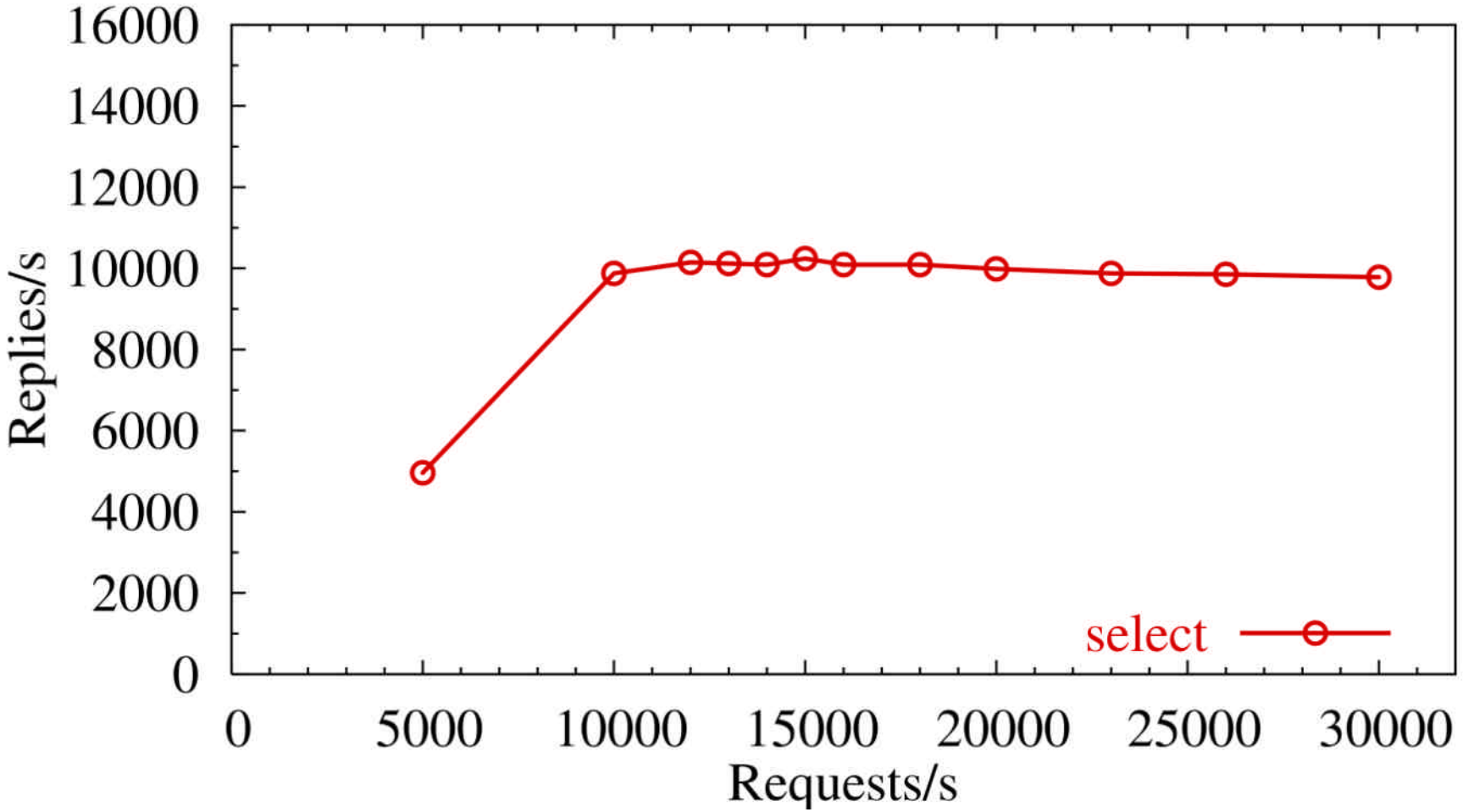
SPECweb99-like Workload w/o idle conns



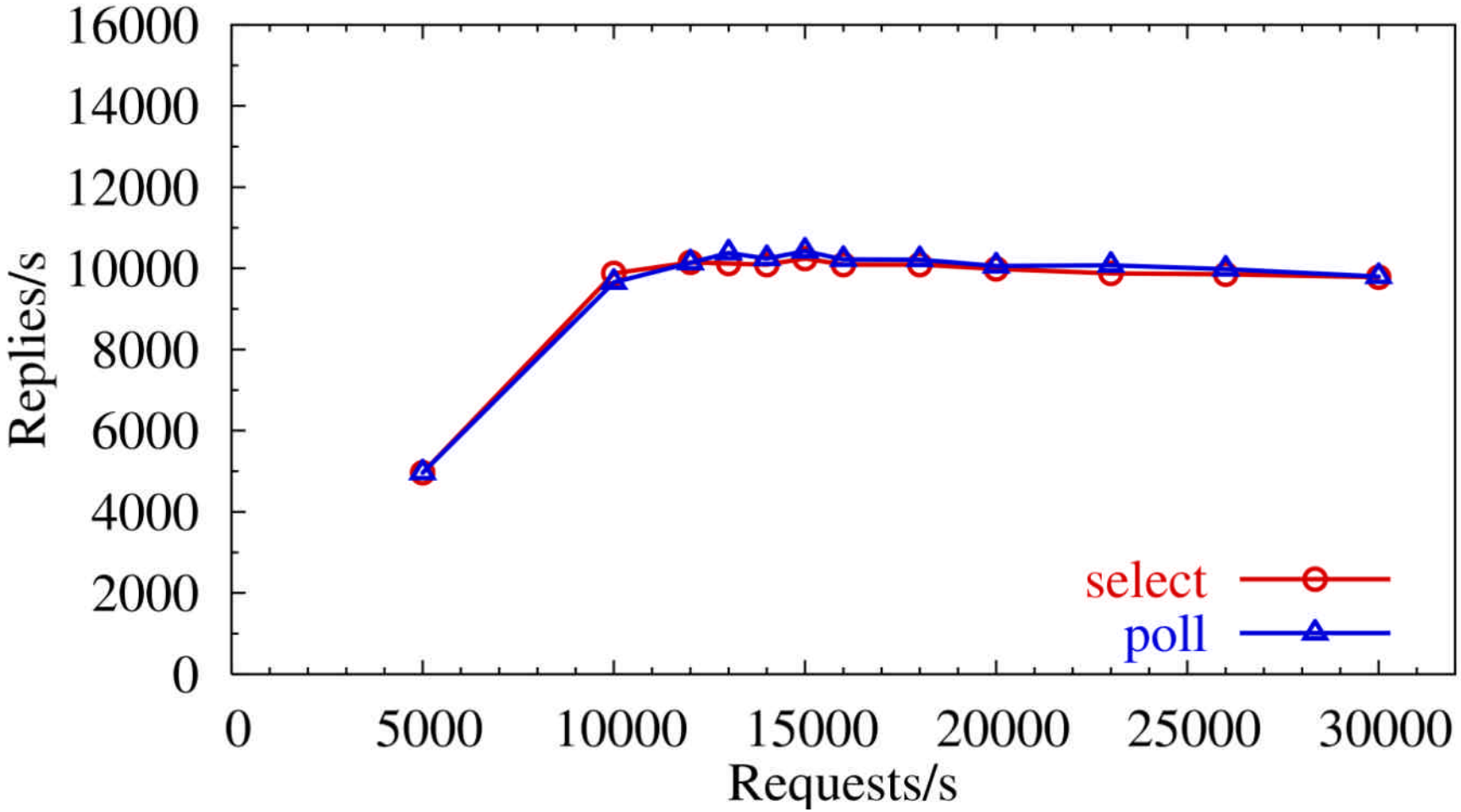
SPECweb99-like Workload w/o idle conns



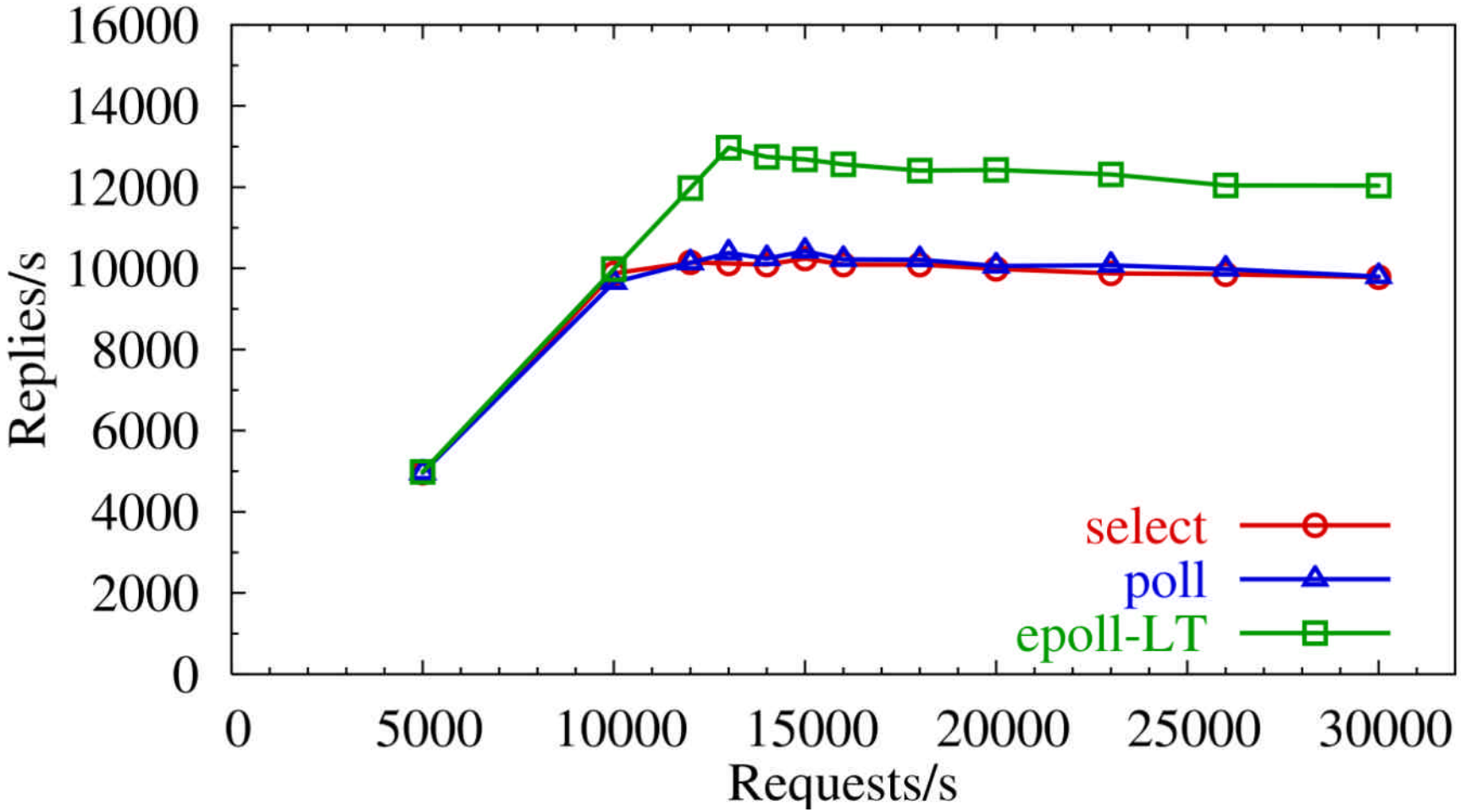
SPECweb99-like: 10,000 idle conns



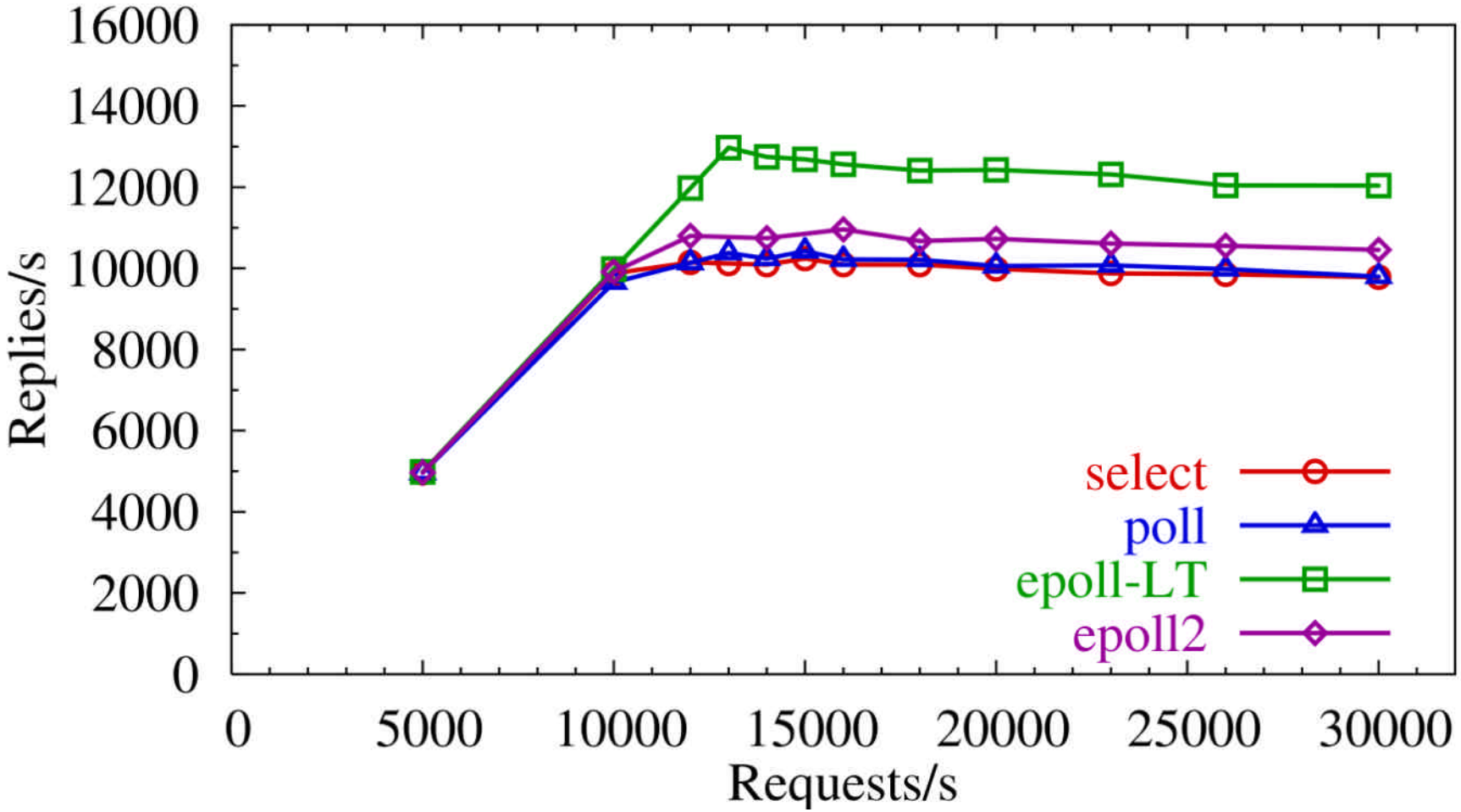
SPECweb99-like: 10,000 idle conns



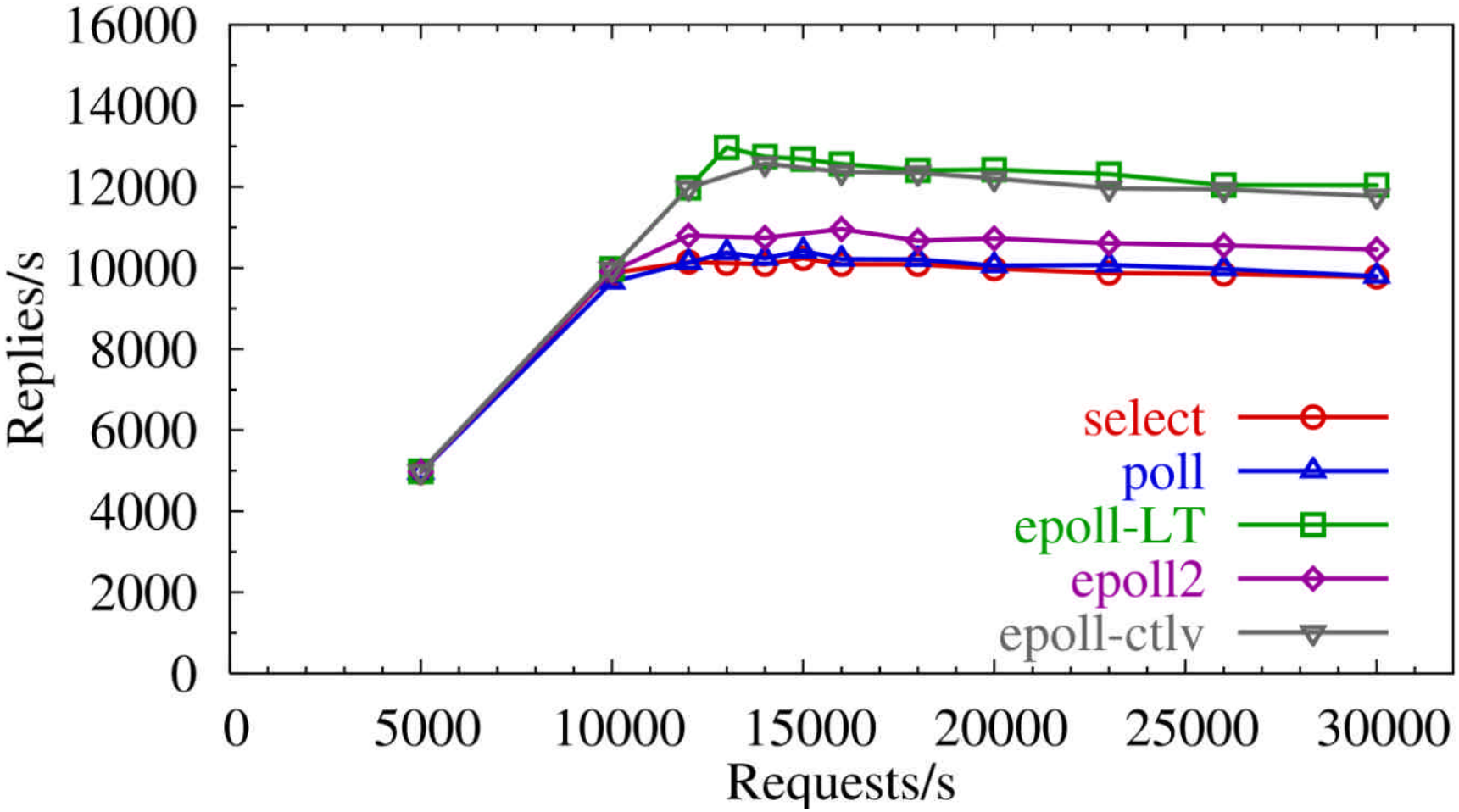
SPECweb99-like: 10,000 idle conns



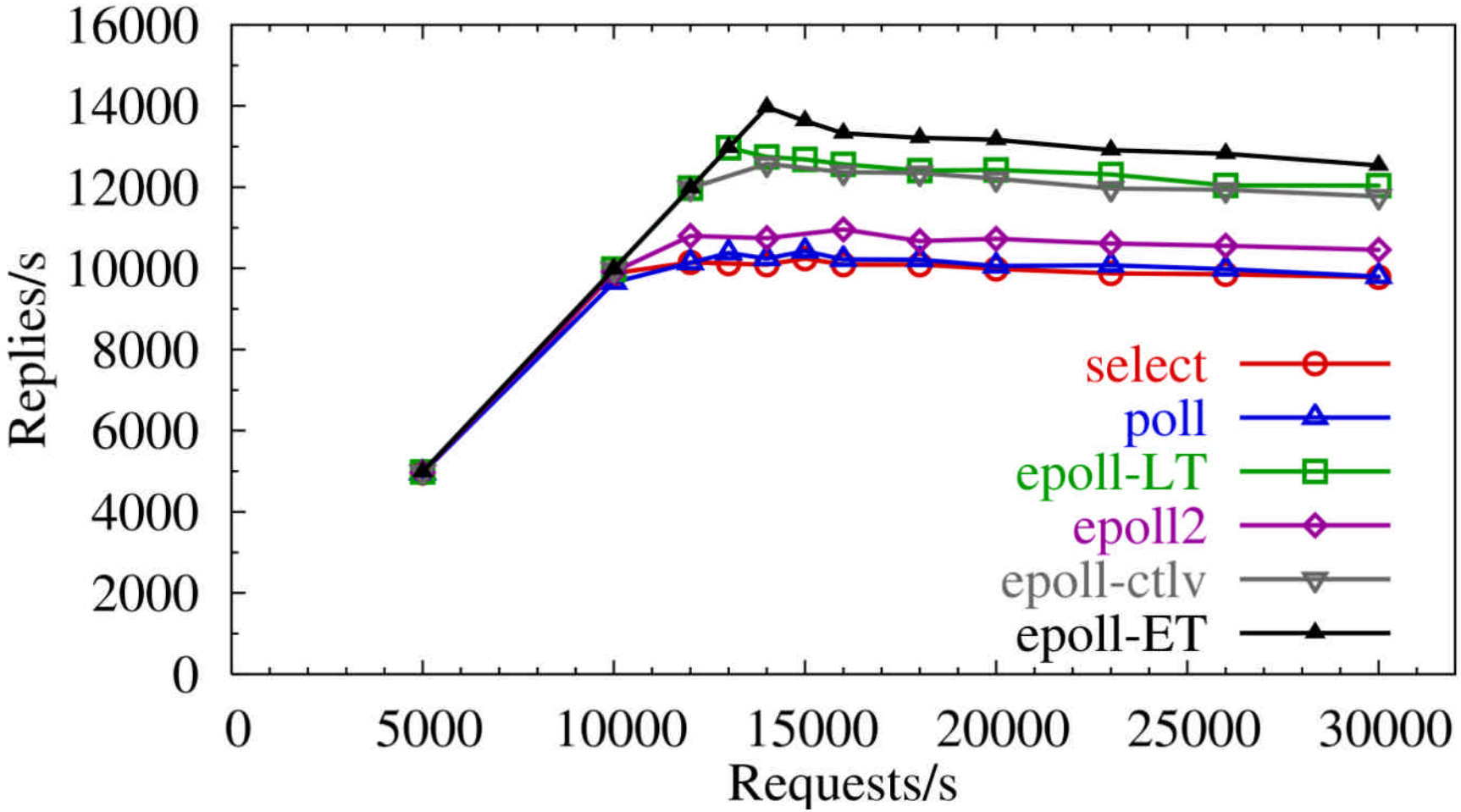
SPECweb99-like: 10,000 idle conns



SPECweb99-like: 10,000 idle conns



SPECweb99-like: 10,000 idle conns



Discussion

- Reducing **`epoll_ctl`** costs did not improve tput
  - **`epoll-LT/ET`** quite similar performance
  - Why not better performance from ET?
- **`select`** and **`poll`** do well w/o idle connections
  - multiple accepts reduce event dispatch overheads
- How realistic is using idle connections?

`http://www.hpl.hp.com/research/linux/userver`

Questions?