Comparing and Evaluating epoll, select and poll Event Mechanisms

Louay Gammo, Tim Brecht, Amol Shukla, and David Pariag

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

accept
read
open/read/close
write/sendfile
close
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

Get Events Phase (figure out what to do)
get events
  select(), poll(), epoll_wait()

Accept Phase
get new connections
  n = accept_loop(accept_limit)

Work Phase
process requests / connections
  read()   write() / sendfile()
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

```c
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);
```

1 10 Inf

Lower Rate accept_limit Higher Rate
accept() strategies

![Graph showing replies per second vs requests per second for limit-1 strategy. The graph shows an increasing trend up to a peak, followed by a sharp decline, and then a more gradual decrease.](image-url)
accept() strategies

[Figure: Graph showing the performance of different 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

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

Changing interest
\[
\text{parray}[i].\text{fd} = \text{sd};
\text{parray}[i].\text{events} \ |= \text{POLLIN};
\text{parray}[i].\text{events} \ &= \text{~POLLOUT};
\]
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_CTL_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)

• httpperf
  – 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)

• **httpperf**
  – 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

22,000 req/s
## gprof comparison

<table>
<thead>
<tr>
<th>Syscall</th>
<th>select</th>
<th>epoll-LT</th>
<th>poll</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>21.51</td>
<td>20.95</td>
<td>20.97</td>
</tr>
<tr>
<td>close</td>
<td>14.90</td>
<td>14.05</td>
<td>14.79</td>
</tr>
<tr>
<td>select</td>
<td>13.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>poll</td>
<td></td>
<td></td>
<td>13.32</td>
</tr>
<tr>
<td>epoll_wait</td>
<td></td>
<td>7.15</td>
<td></td>
</tr>
<tr>
<td>epoll_ctl</td>
<td></td>
<td>16.34</td>
<td></td>
</tr>
<tr>
<td>setsockopt</td>
<td>11.17</td>
<td>9.13</td>
<td>10.68</td>
</tr>
<tr>
<td>accept</td>
<td>10.08</td>
<td>9.51</td>
<td>10.20</td>
</tr>
<tr>
<td>write</td>
<td>5.98</td>
<td>5.06</td>
<td>5.70</td>
</tr>
<tr>
<td>fcntl</td>
<td>3.66</td>
<td>3.34</td>
<td>3.61</td>
</tr>
<tr>
<td>sendfile</td>
<td>3.43</td>
<td>2.70</td>
<td>3.43</td>
</tr>
</tbody>
</table>
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**
  - 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

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

Values shown are percentage of time spent
## gprof results @ 22,000 req/s

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

Values shown are percentage of time spent
### gprof results @ 22,000 req/s

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

Values shown are percentage of time spent
**gprof results @ 22,000 req/s**

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

Values shown are percentage of time spent
## gprof results @ 22,000 req/s

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

Values shown are percentage of time spent
# gprof results @ 22,000 req/s

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

Values shown are percentage of time spent
gprof results @ 22,000 req/s

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

Values shown are percentage of time spent
1 Byte File: 10,000 idle conns

![Graph showing Replies/s vs Requests/s for epoll-LT](image-url)
1 Byte File: 10,000 idle conns

Replies/s

Requests/s
1 Byte File: 10,000 idle conns

![Graph showing the performance of epoll-LT, epoll2, and epoll-ctlv in terms of replies per second with varying requests per second.]
1 Byte File: 10,000 idle conns

- epoll-LT
- epoll2
- epoll-ctlv
- epoll-ET

Replies/s vs Requests/s graph with different line styles and markers for each epoll type.
1 Byte File: 10,000 idle conns

- epoll-LT
- epoll2
- epoll-ctlv
- epoll-ET
- poll

Replies/s vs Requests/s graph showing performance comparison among various file operations.
1 Byte File: 10,000 idle conns

![Graph showing performance metrics for different file operations]

- epoll-LT
- epoll2
- epoll-ctlv
- epoll-ET
- poll
- select
SPECweb99-like Workload w/o idle conns

![Graph showing the relationship between replies per second and requests per second for a SPECweb99-like workload without idle connections.]
SPECweb99-like Workload w/o idle conns

![Graph showing replies per second vs. requests per second for 'select' and 'poll' methods. The graph illustrates the performance comparison between the two methods under varying request loads.]
SPECweb99-like Workload w/o idle conn

![Graph showing performance of different methods with Requests/s and Replies/s]
SPECweb99-like Workload w/o idle conns

![Graph showing the performance of different methods (select, poll, epoll-LT, epoll2) in terms of replies per second versus requests per second. The graph highlights the results under a SPECweb99-like workload without idle connections.](image)
SPECweb99-like Workload w/o idle conns
SPECweb99-like Workload w/o idle conns

![Graph showing performance comparisons for different methods like select, poll, epoll-LT, epoll2, epoll-ctlv, and epoll-ET.]
SPECweb99-like: 10,000 idle conns
SPECweb99-like: 10,000 idle conns
SPECweb99-like: 10,000 idle conns

- select
- poll
- epoll-LT
SPECweb99-like: 10,000 idle conns
SPECweb99-like: 10,000 idle conns
SPECweb99-like: 10,000 idle conns

![Graph showing performance comparison for different methods](image-url)
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?