occwserv: An occam Web-Server

(version 2)

Fred Barnes (frmb2@ukc.ac.uk)

Computing Laboratory, University of Kent,

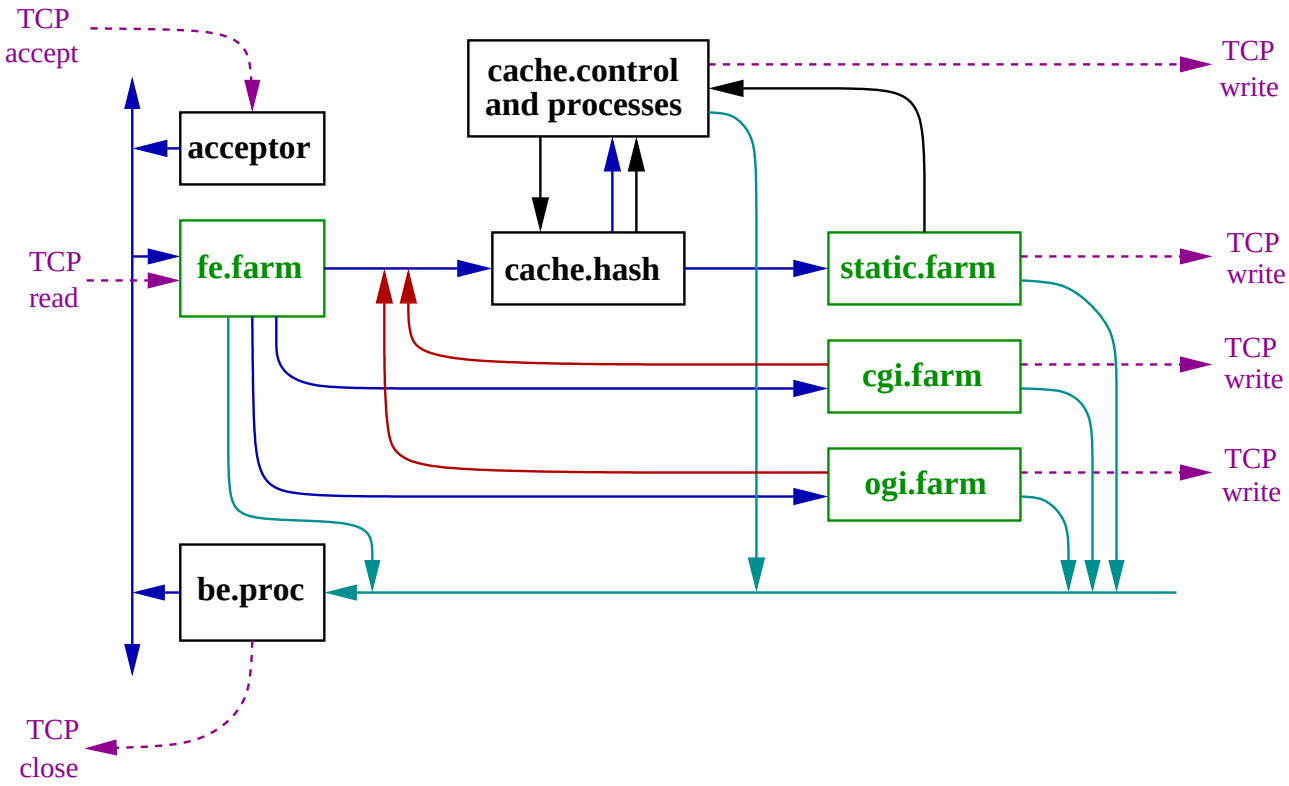
Canterbury, Kent. CT2 7NF

Contents

Introduction

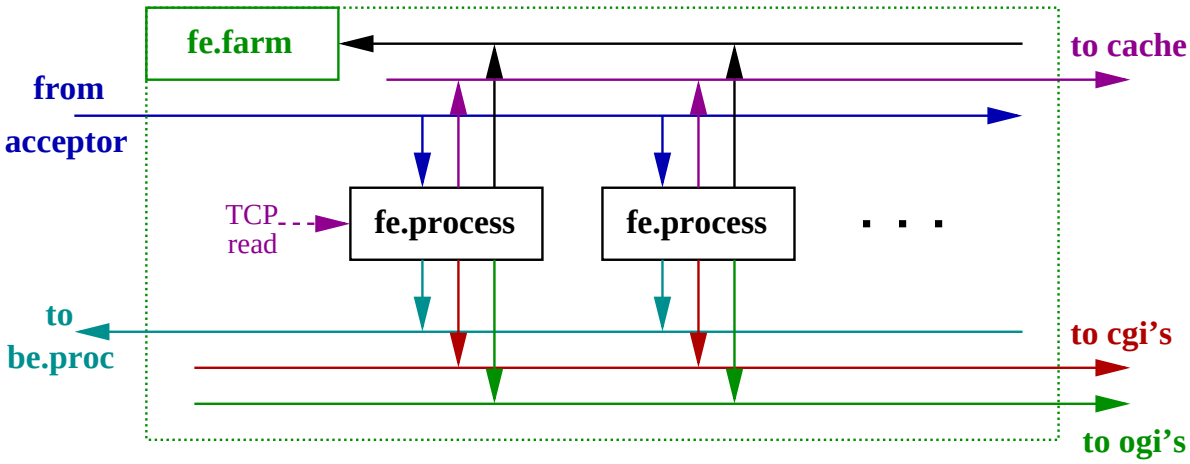
- Web-servers are naturally concurrent:
  - need to handle multiple connections
  - and fairly, ideally
- CSP design:
  - verifiable
  - scalable
- Dynamic occam implementation:
  - implementation correctness
  - performance

Design



- New connections originate in the ‘acceptor’
- Requests read inside the ‘fe.farm’
- Responses generated in various places
- Connections finish in ‘be.proc’

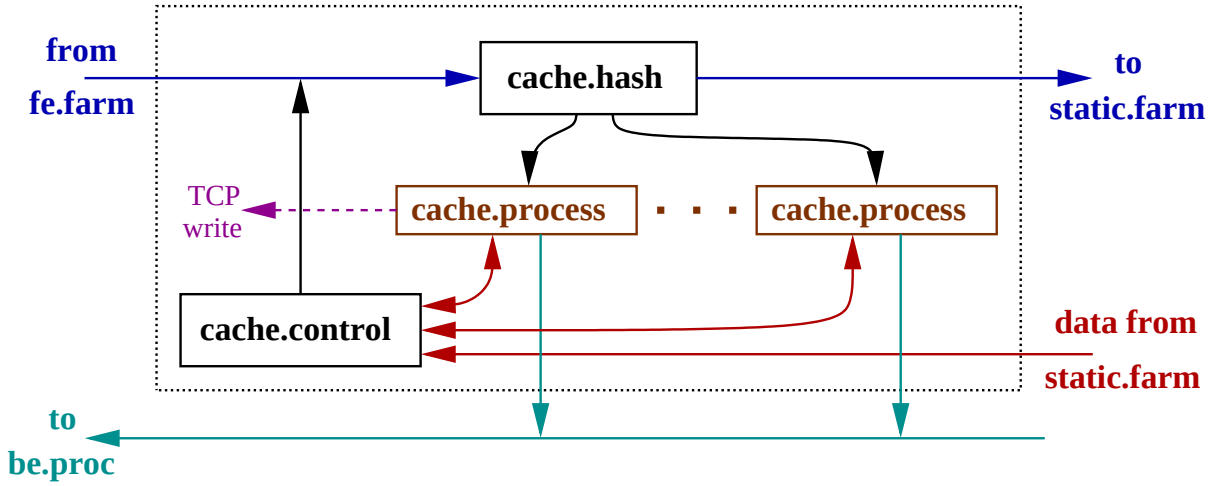
Front-End Farm



- 'fe.farm' maintains a pool of processes
  - workers send −1 when busy
  - and +1 when idle

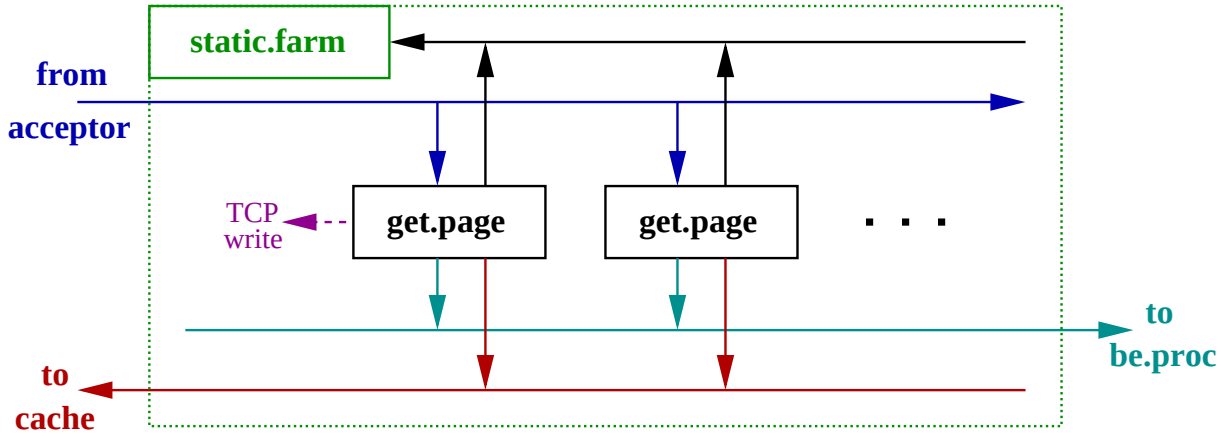
- Each 'fe.process' handles a single client:
  - read the request
  - forward connection (based on request)

Page Caching



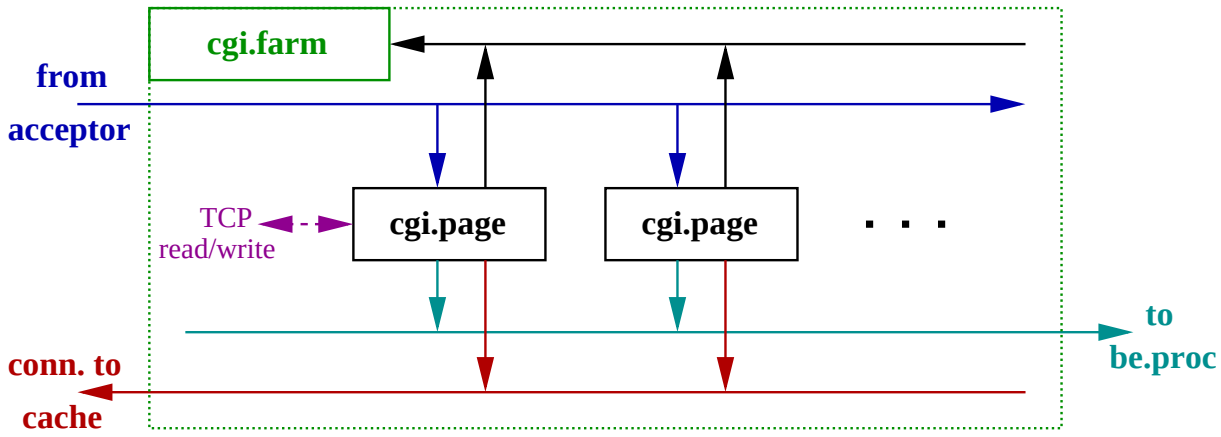
- 'cache.control' handles management
- Requests are hashed and re-directed if a 'cache.process' exists for them
- Non-cached requests are passed to the 'static.farm'
- ... that updates 'cache.control' after pages have been retrieved

Static Pages



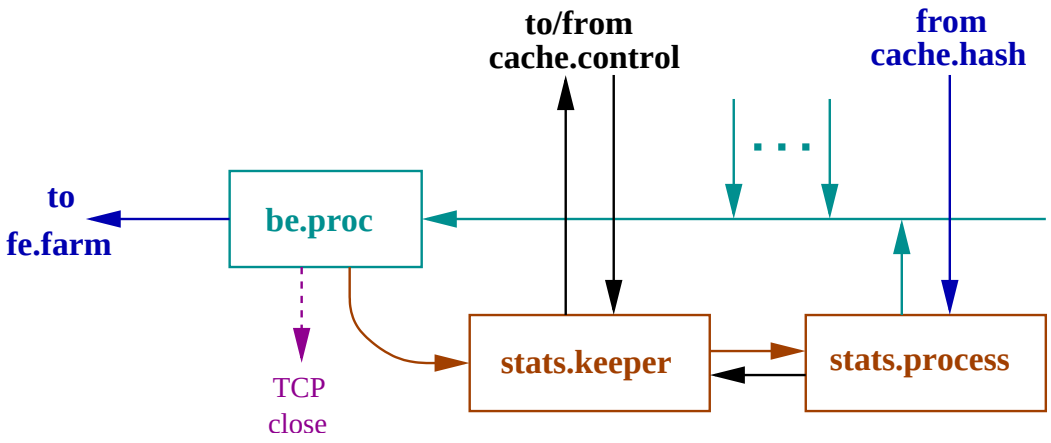
- As before, a pool of at least n free workers is maintained
- The 'get.page' process copies file contents to the client
  - using the 'sendfile' system-call
  - after sending the headers

CGI Scripts



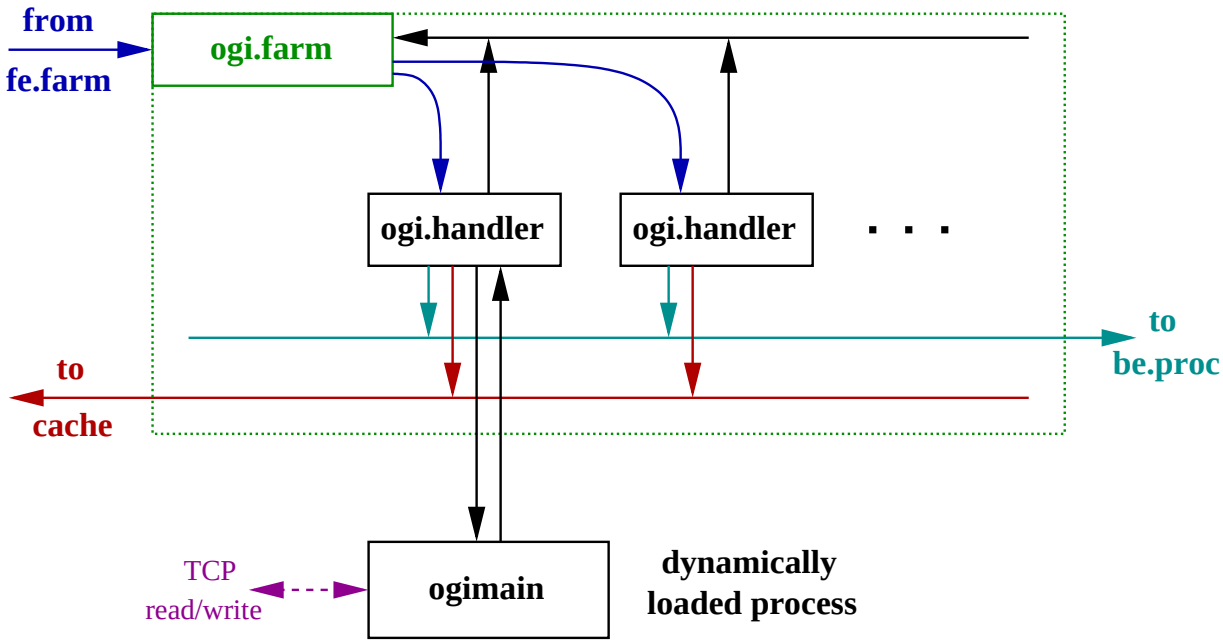
- Follows the design of ‘static.farm’
- The ‘cgi.page’ process executes the specified script, sending output directly to the client

Back-end Processing



- Primarily responsible for closing or recycling client connections
- Also reports connection statistics to the 'stats.keeper' process

The occam Gateway Interface

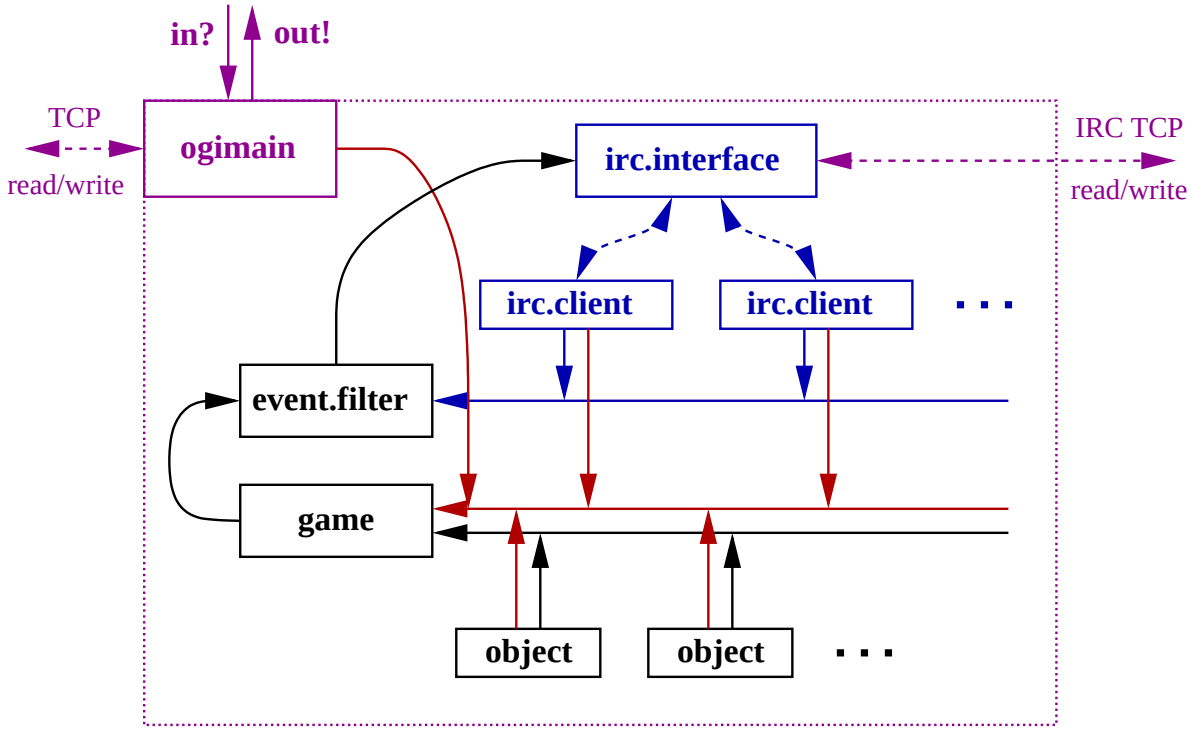


- OGI modules are dynamically loaded
- Connections are serialised in 'ogi.farm'
- OGIs may handle $>$ 1 client simultaneously
  - simple setup protocol required to do this correctly

OGIs

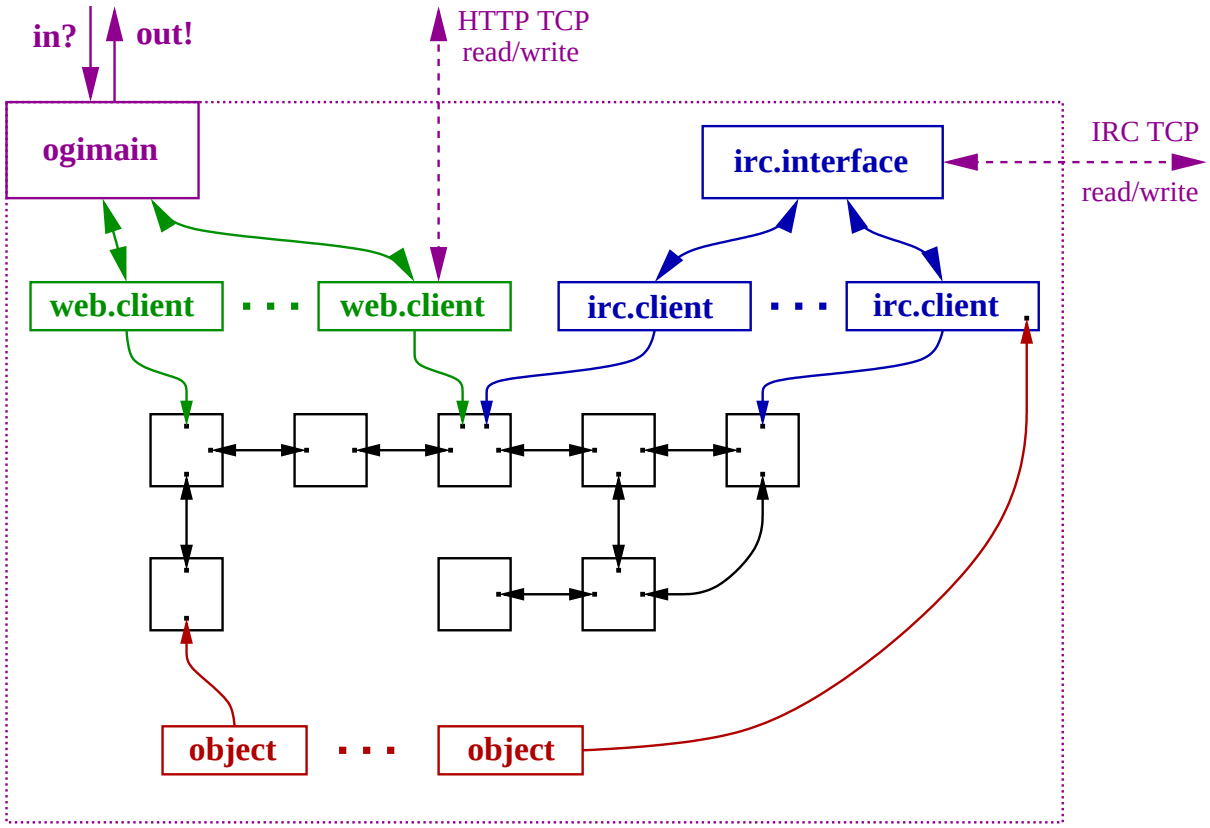
- OGIs can be one-shot or persistent
  - removing the OS-process startup/shutdown cost associated with CGI scripts
- Since an OGI may handle multiple clients, interactions between clients:
  - are simple to implement
  - and are controllable
- Simple web-based 'chat' OGI, for example
- ... or something more complex

The occam Adventure

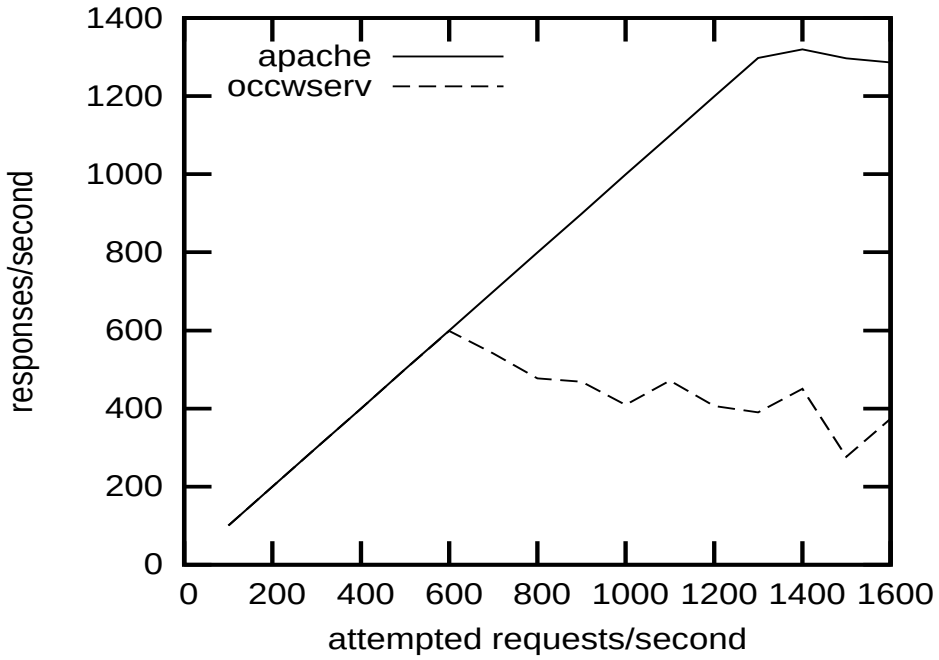


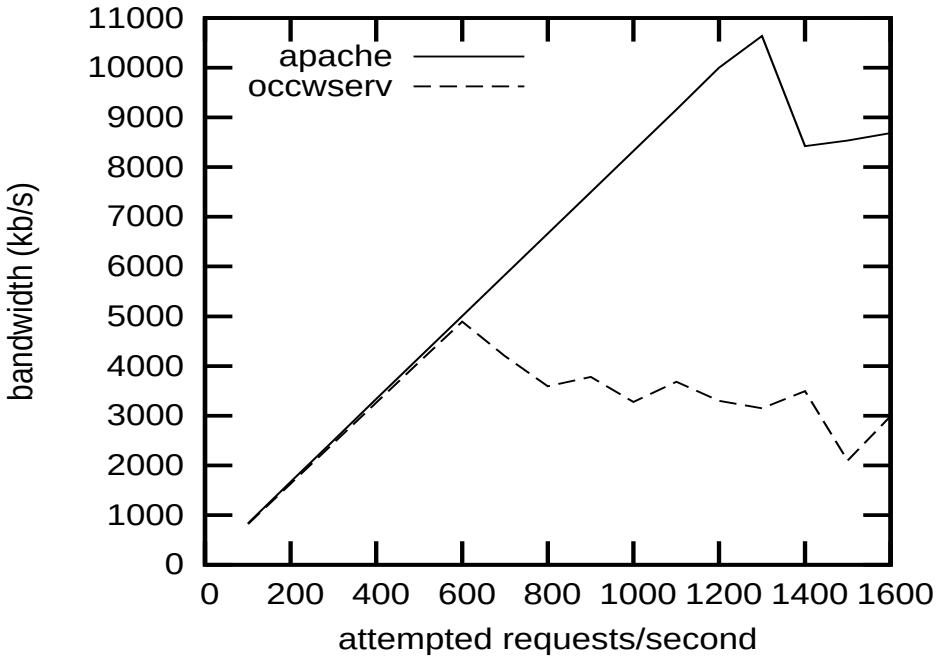
- Supports an IRC interface in addition to the web-interface
  - Creates a 'bot' that interacts with users
- Adding a traditional MUD 'telnet' interface would be trivial :-)

A New Adventure



- Decentralised state, cleaner design
- Channel-ends move around the network

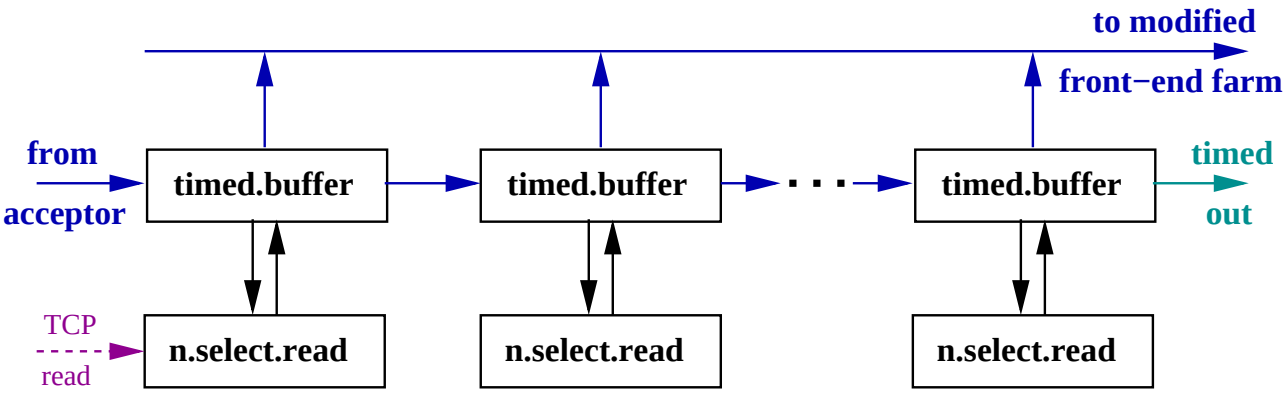

1400
1200
1000
800
600
400
200
0
responses/second
apache
occwserv
0 200 400 600 800 1000 1200 1400 1600
attempted requests/second


11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
bandwidth (kb/s)
apache
occwserv
0 200 400 600 800 1000 1200 1400 1600
attempted requests/second

Deficiencies

- Performance is limited
  - not an ideal benchmark
  - server was run with full debugging
- Bottleneck from blocking system-calls
  - collect/dispatch time is significant
  - frequent (Linux) rescheduling
- Each client request requires at least three blocking calls for a request
  - could do something more intelligent in the front-end farm

Improving Performance



- Each 'timed.buffer' process holds a number of connections
  - inactive connections move left to right
  - active connections stay put
- Reduces the number of blocking calls made
- ... or get a faster PC..! :-(

On-going Research

- Although performance is *currently* limited
  - design simplicity and correctness count for a lot
  - no buffer overflow potential, zero aliasing, zero race-hazard, ...
- Blocking syscalls are being 'upgraded'
  - including a new Linux device-driver to significantly improve performance
  - the 'cspdriver'
- Raw-metal web-serving (with RMoX)

Accessing the Server

`http://wotug.kent.ac.uk/ocweb/`

- Currently off-line while I move offices
- Should be back around the 20th Sept.
- Hope to have performance issues resolved in a month or two.. :-)