Evaluating Unikernels

Ian Briggs
Matt Day
Yuankai Guo
Peter Marheine

CS 6480: Advanced Computer Networks
December 3, 2014

Introduction

Unikernels

- What is a Unikernel?
  - A specialized OS kernel linked with app code+data
    - e.g. TCP/IP stack + threading + HTTP server + HTML files
  - Easily loadable into today's virtualized cloud
  - Advantages:
    - smaller memory footprint
    - some unikernels use a high-level language for everything
    - smaller attack footprint
  - Disadvantages:
    - this stuff is new: very buggy, few users
    - (Ian has a tale of woe)

Unikernel stack

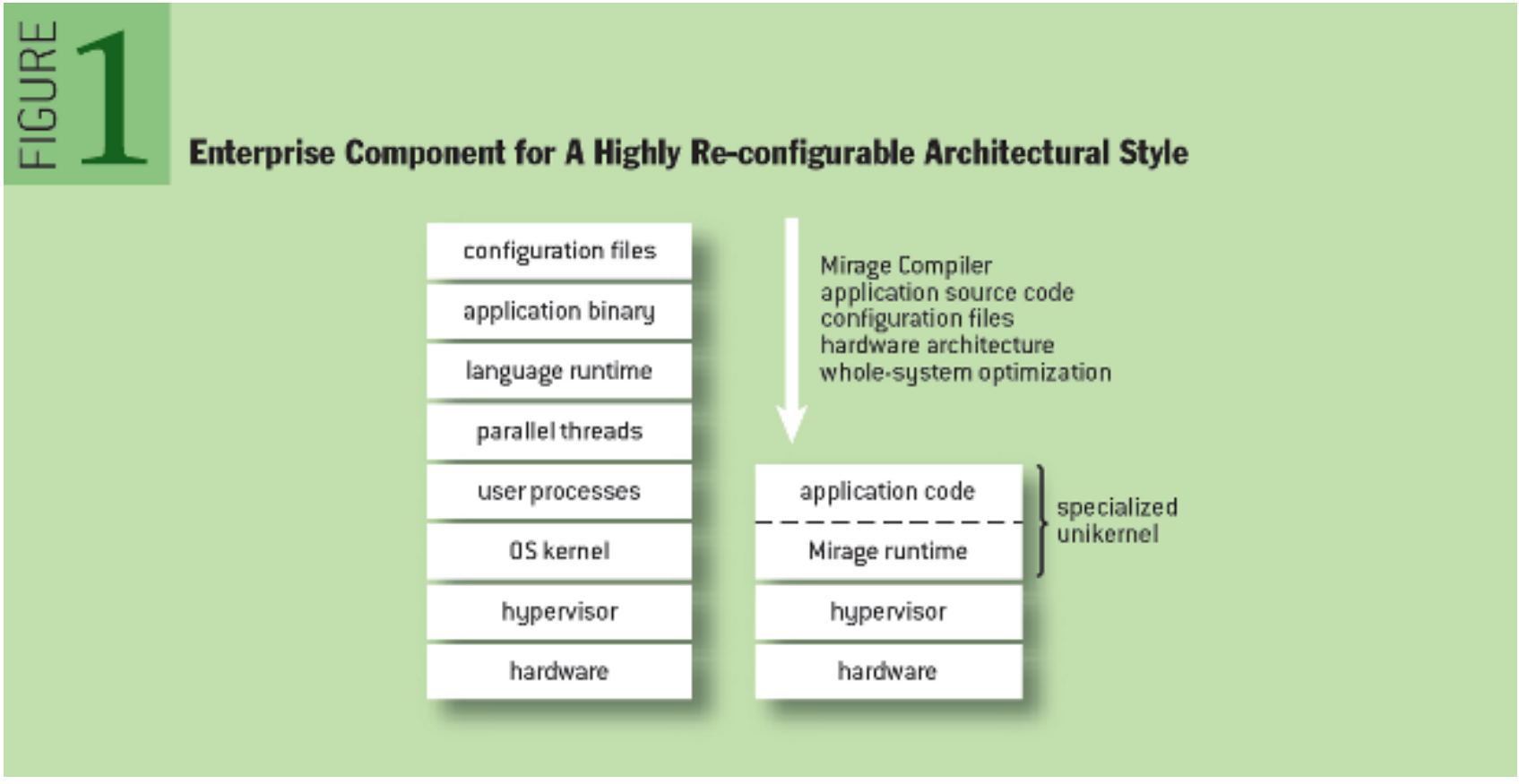

Figure 1: Traditional vs. Unikernel

Related Work

- This is the first survey of unikernels (that we could find)
- Each specific unikernel paper has its own results
  - But, the code used to obtain the Mirage results was accidentally deleted!

Platforms Evaluated

- OSv: specialized BSD UNIX kernel linked with existing apps
- Mirage OS: OCaml everywhere
- LAMP: Ubuntu Linux, BIND DNS server, Apache HTTP server

Performance Model

Performance Factors

- What affects the performance?
  - CPU time
  - Memory space
  - Network bandwidth

Figure 2: Factors of server performance

CPU Time

- The dominant factor
- The longer the CPU time, the lower the performance
- Influence the response time

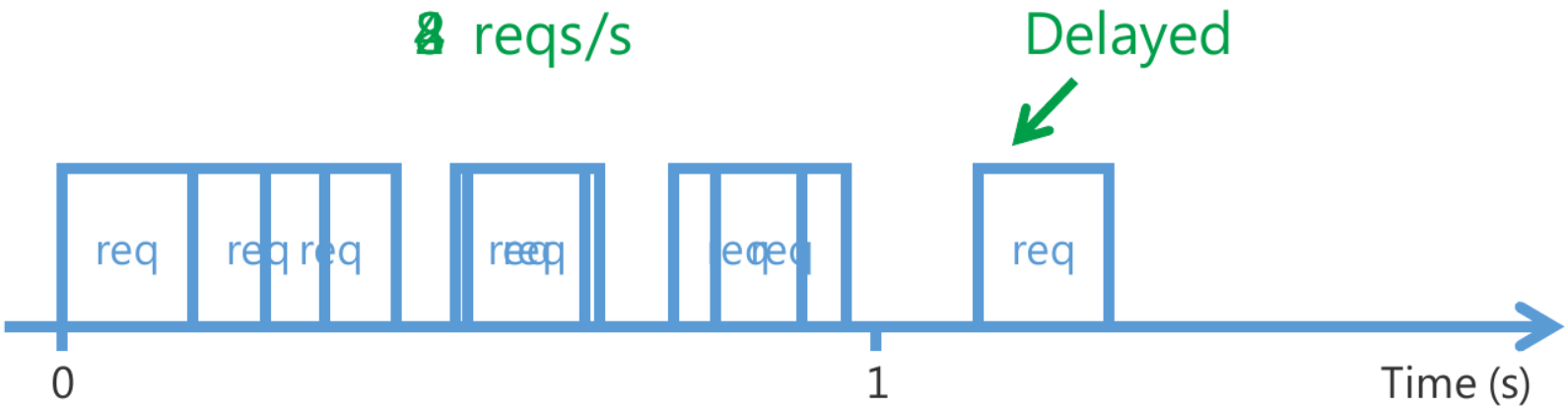

Figure 3: CPU-constrained performance

Memory Space

- Determines the amount of concurrent requests
- Performance drops when page swapping happens
- Longer response time when waiting for free memory
- Deadlocks may occur in extreme situations

Bandwidth

- Limits the speed of concurrent connections

Testing Methodology

How to measure performance?

- How to drive a service to its maximum performance?
- How to judge the service has reached the maximum performance?
- How to measure it quantitatively?

The Solution

- Send requests in varying rates, e.g. from 100 to 10,000 reqs/s
- Monitor the response rate and time

How It Works

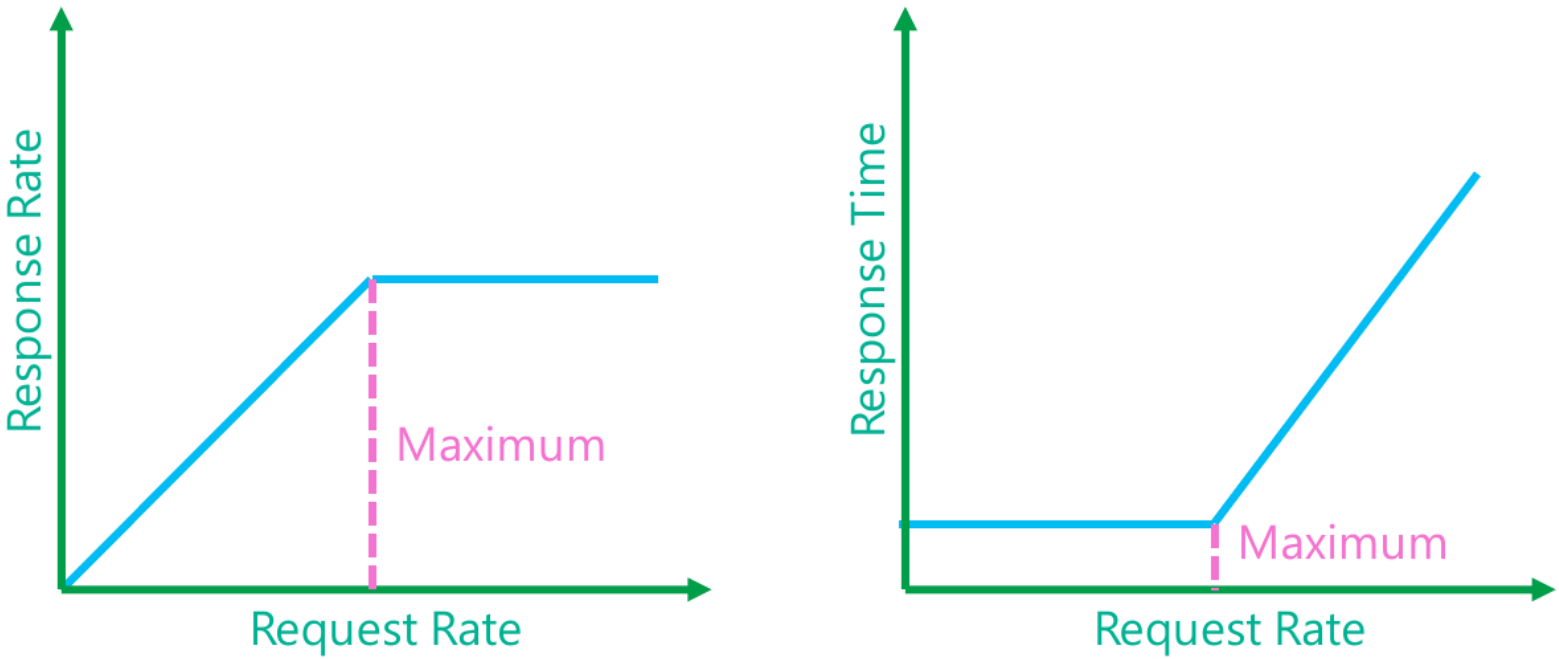

Figure 4: Request rate against measured parameters

Testing Tools

httperf & BIND queryperf

- Standard measurement tools used by other researchers
- Send HTTP or DNS requests with adjustable rate
- Output statistical data, including response rate and time

Automatic Testing Framework

- Standardize and automate the testing, achieving repeatability
- Run httperf and queryperf with varying request rate
- Collect data from the output
- Do plotting to visualize the data

Example Result

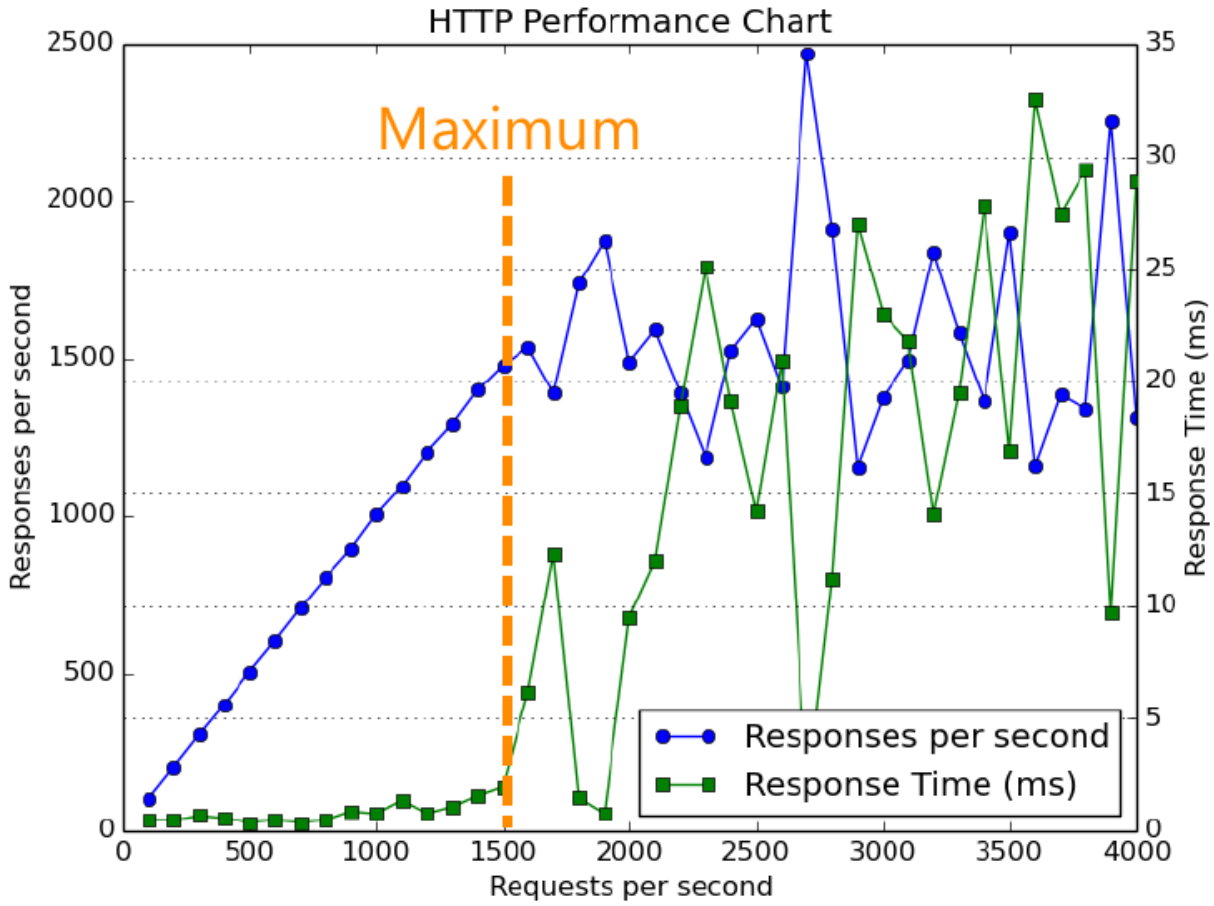

Figure 5: HTTP server from 100 to 4000 req/s

Testing Hardware

- We utilized Emulab
- Two nodes connected with a gigabit link
- All tests used 1 CPU core, giving advantage to unikernels
- Unless stated one gigabyte of RAM was used
- Stats
  - d710 machines
  - 64-bit Intel Quad Core Xeon E5530
  - 12 GB of RAM

Mirage

Architecture

Appeal

- OCaml
  - garbage collected
  - statically typed
- Can compile to Unix for development and to Xen unikernel for deployment

Reality

- Garbage collection halts system for long periods of time
- Dual compilation needs special work on the programmer's part
- Unix and Xen compilations often don't work similarly

Mirage Porting

- Mirage uses special interfaces which require total rewrites of existing OCaml
- Mirage itself is missing functionality
  - Only FAT32 filesystem is supported
  - Block caching must be provided by the programmer
  - As of July this year delete would corrupt the filesystem
- Existing codebases written for Mirage degrade quickly
  - No backward compatibility

Implementations Used

- Best Mirage code out there
- Basically toy implementations
  - Hard coded IP addresses and ports
  - TCP/IP library prints to screen on disconnect

Flaps

Mirage

- Mirage went through multiple versions during the semester
  - Breaking working code
  - Breaking package manager required
  - Changing front end interface used

Flaps

httperf

- httperf would report long round trip times
  - During those times the server could be used with no problem
  - Failure point was not consistent with rate
  - This would manifest on all implementations

As it turns out this is because the sockets were being closed by httperf and held in a TIME_WAIT state

OSv

OSv Configuration

Architecture

- Linux ABI
- All threads share an address space
- `clone()` but no `fork()`

Implications

- Low-cost context switches
  - No TLB reload
  - No kernel/userspace copies
- Risk of memory corruption

OSv Porting

- Build with Linux toolchain
- Compile to shared object (`-fPIC -shared`)
- Include shared libs (`ldd`) in appliance

iperf

- Already ported
- Includes zero-copy I/O

HTTP

- lighttpd
- `CFLAGS=-fPIC ./configure`
- Manual link phase for .so
- OSv hang under Xen- run in KVM

DNS

- ISC BIND9
- OSv patches required
  - `syslog(3)` stub
  - Reexport glibc symbols
  - Ignore unsupported socket features
- Post-configure `Makefile` patches

```
int linux_to_bsd_msghdr(struct msghdr *hdr) {
    /* Ignore msg_control in OSv */
    hdr->msg_control = NULL;
    hdr->msg_controllen = 0;
    hdr->msg_flags =
        linux_to_bsd_msg_flags(hdr->msg_flags);
    return (0);
}
```

Evaluation

iperf

Not captured.

- Links saturated.
- CPU use not yet measured.
  - Expect OSv to be more efficient

DNS

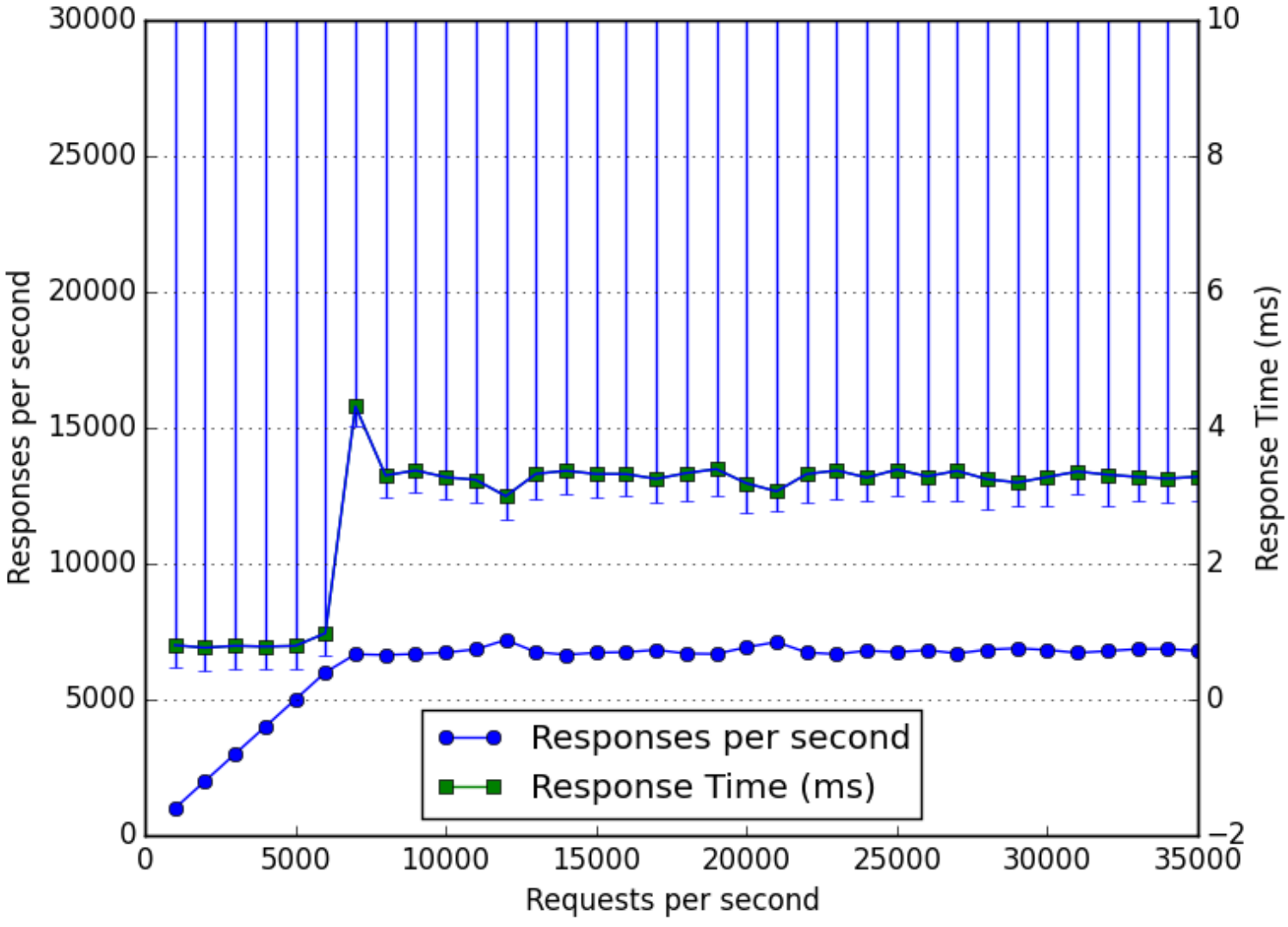

Figure 6: Mirage DNS performance

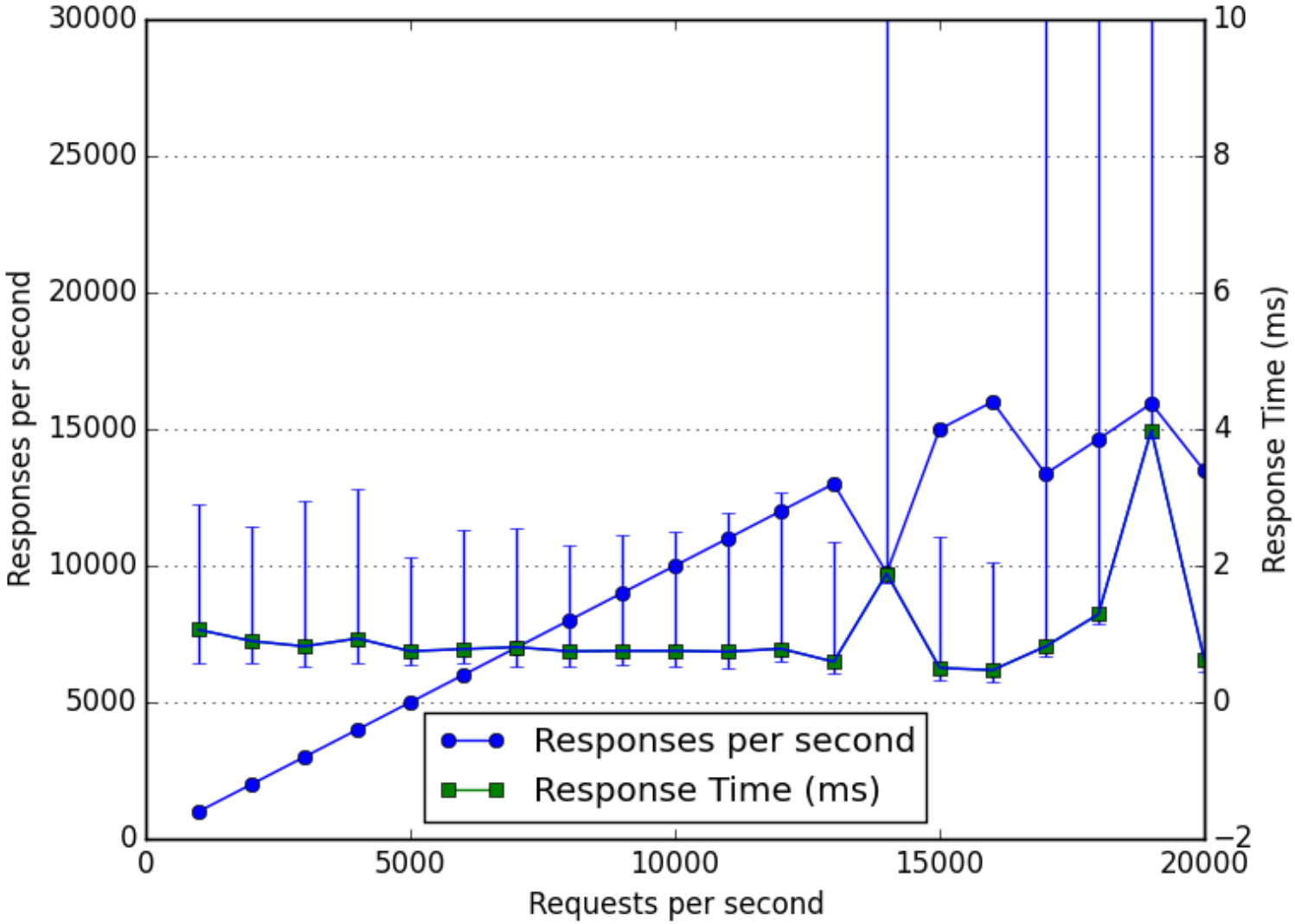

Figure 7: Linux DNS performance

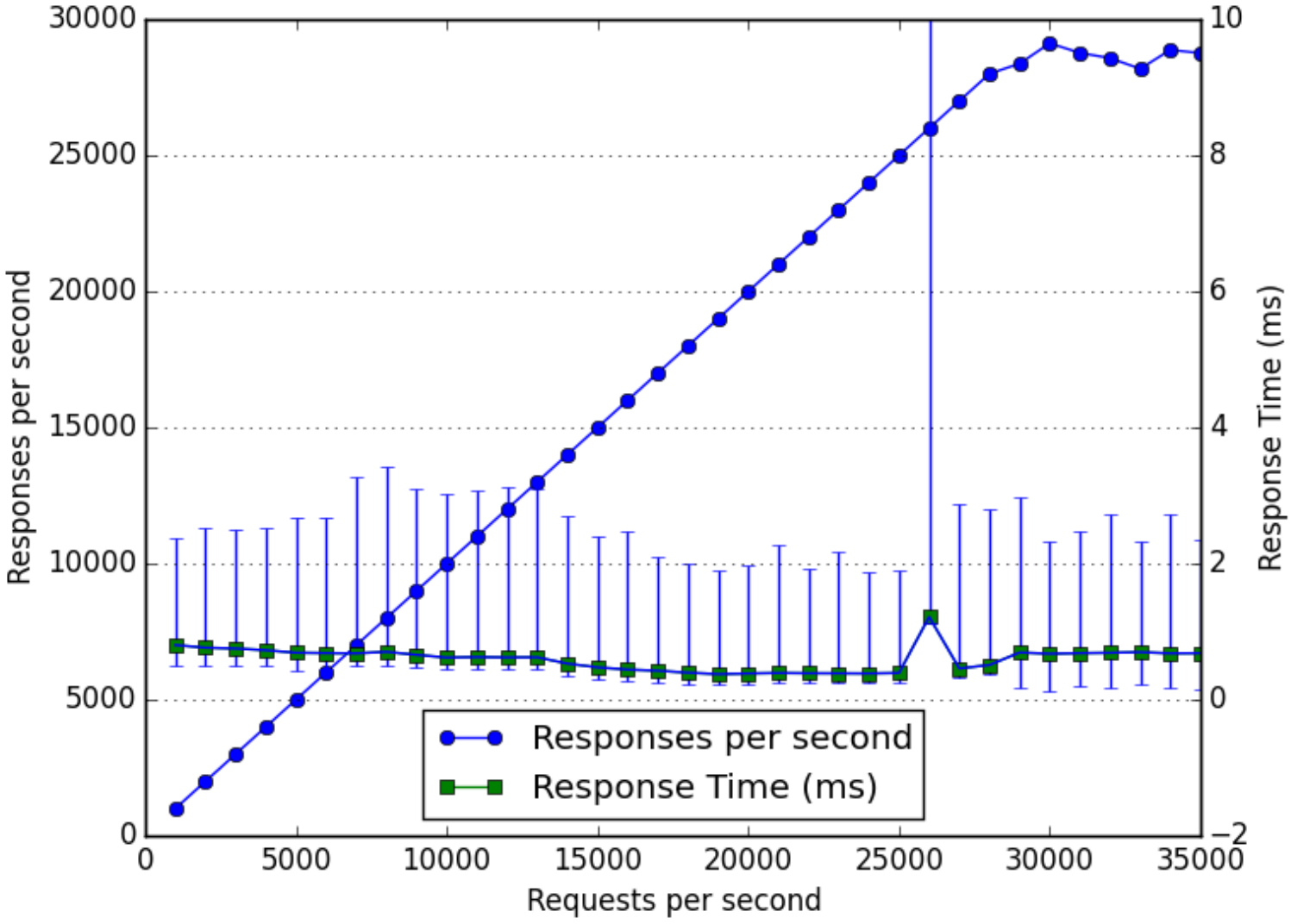

Figure 8: OSv DNS performance

HTTP

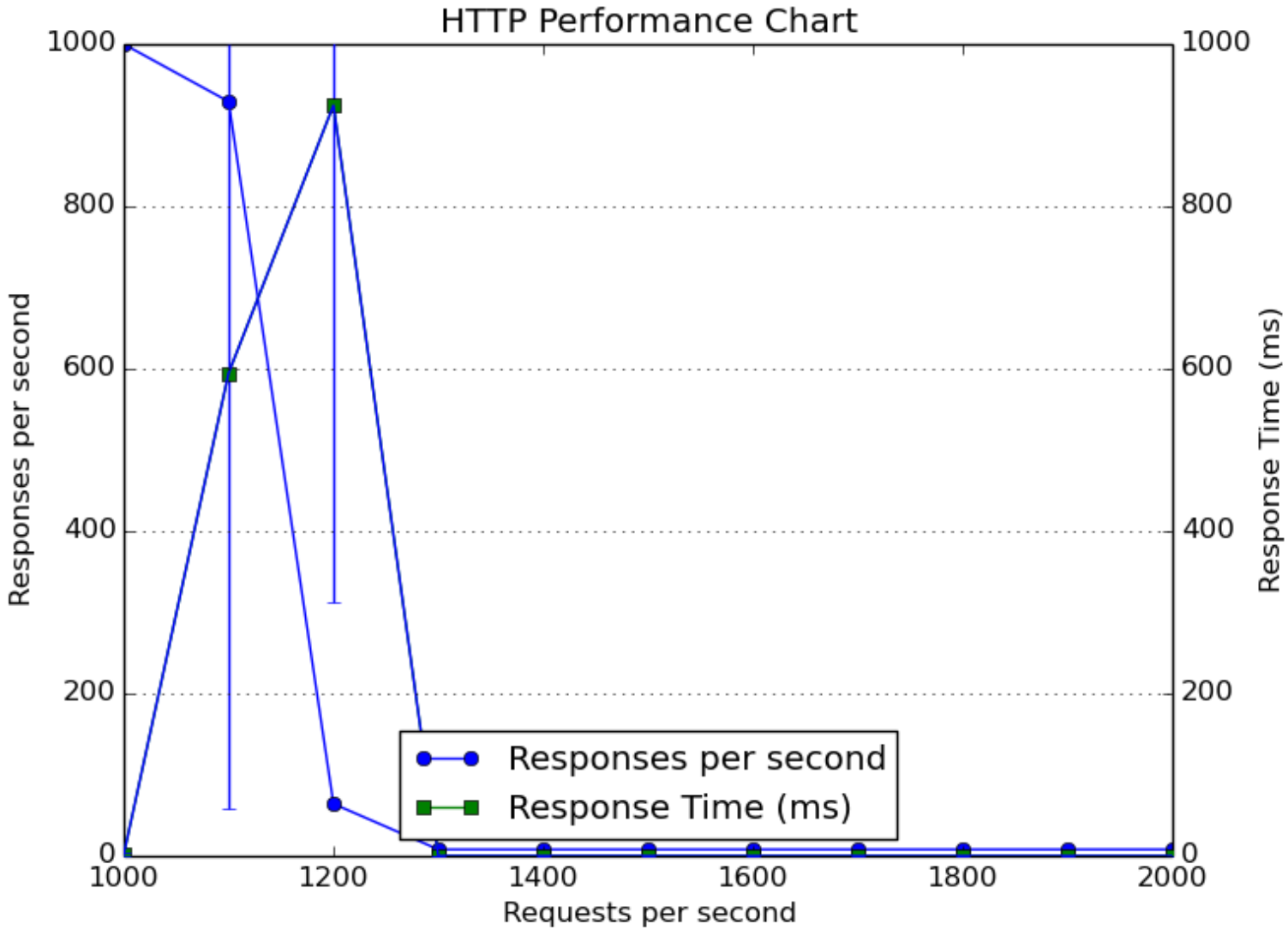

Figure 9: Mirage HTTP performance

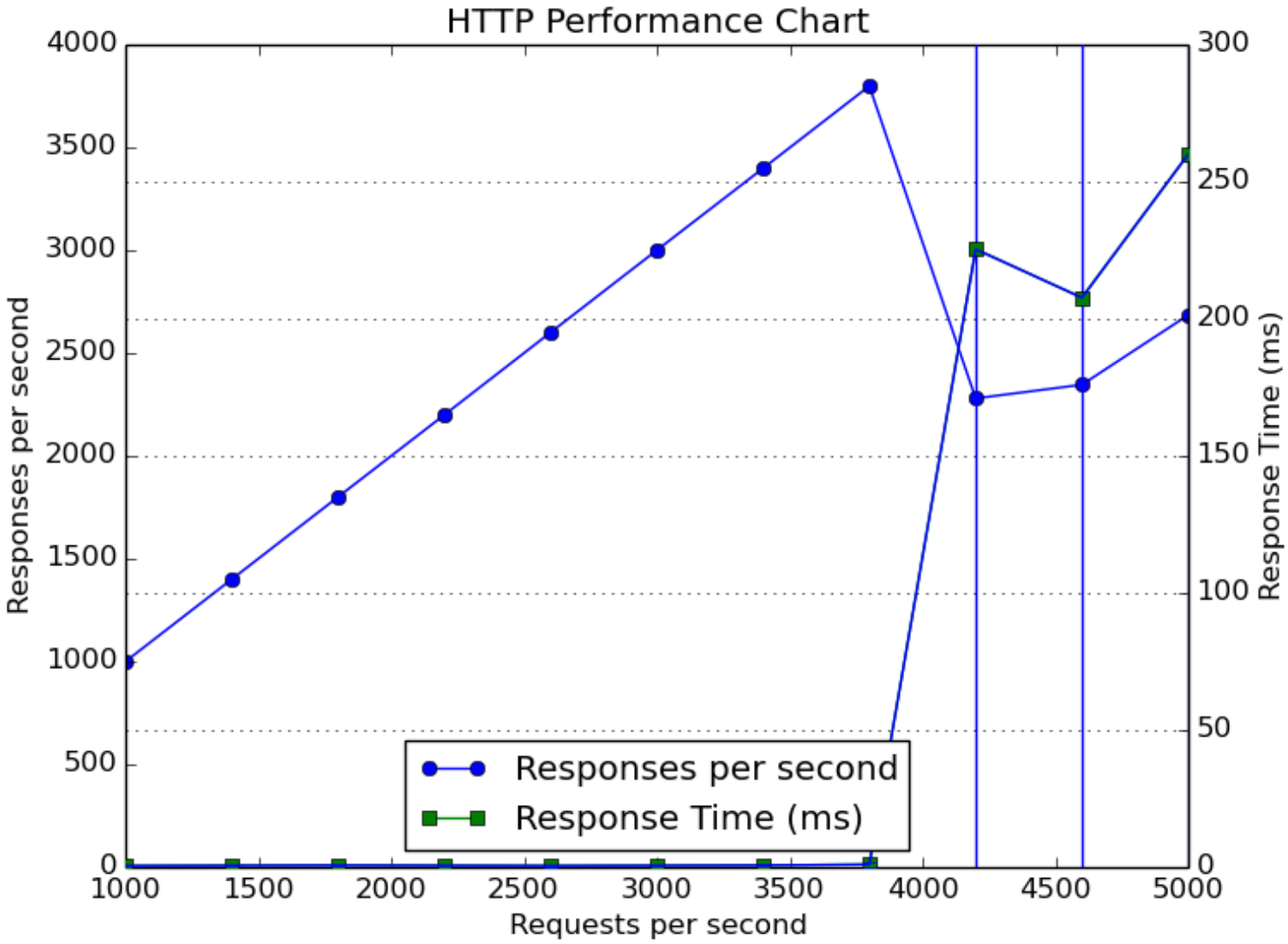

Figure 10: Linux HTTP performance

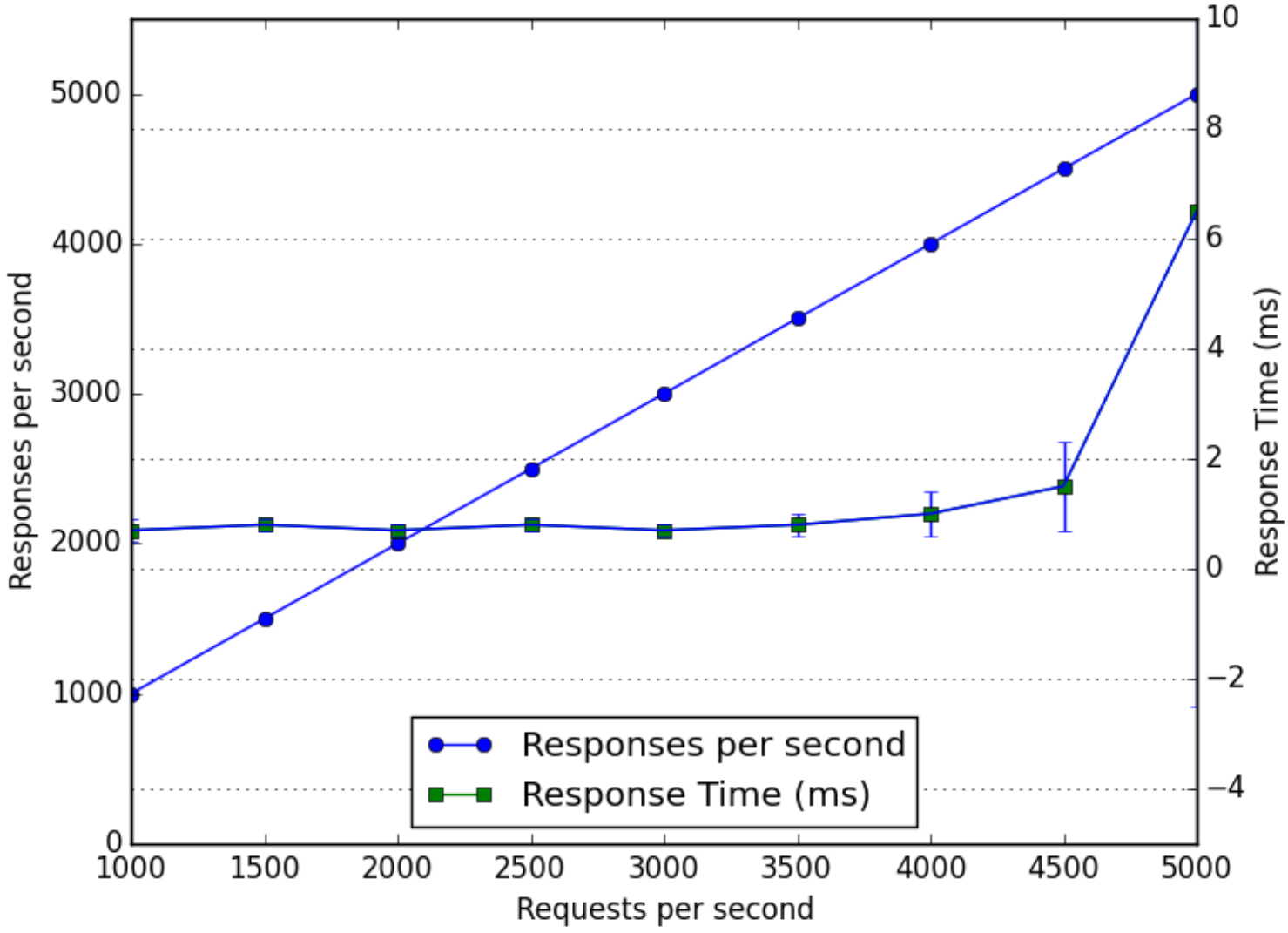

Figure 11: OSv HTTP performance

Further Work

- Other systems (Docker, Erlang on Xen?)
- Improve HTTP client
- Test a dynamic application
- Fix platform bugs..

Questions?

- We have been:
  - Evaluating Unikernels
    - Ian Briggs
    - Matt Day
    - Yuankai Guo
    - Peter Marheine