# A Performance Evaluation of Unikernels*

Ian Briggs, Matt Day, Yuankai Guo, Peter Marheine, Eric Eide†
*School of Computing, University of Utah*

## Abstract

The unikernel architectural model is a relatively recent development, adopted by several new computing platforms, which claims improved performance in cloud applications by eliminating unnecessary software components from virtual machines. To validate these published claims and explore the suitability of two existing unikernel implementations, Mirage OS and OSv, we present a set of realistic macrobenchmarks, using established network performance measurement tools, that compare the performance of standard network applications against a non-unikernel Linux system. Although the performance data produced by these macrobenchmarks suggest that the unikernel model indeed offers performance improvements, we believe our experience shows that the tested unikernel platforms are not yet ready for deployment in production environments.

## 1 Introduction

With the recent rise of cloud computing, virtualization has become a common method for service providers to provide resource isolation across shared platforms with opaque resource assignment [2]. Significant effort has gone into building performant and secure hypervisors [3], seeking to provide performance as near to an unvirtualized environment as possible while hiding the true nature of the hardware platform from applications.

Many services are deployed as single applications running on dedicated virtual machines, themselves running on a shared cloud platform. Observing that these standalone applications do not require many of the services usually provided by an operating system, the concept of a *unikernel* [15] has emerged as a complement to library operating systems or exokernels, in which a single application is compiled into a standalone virtual machine optimized to run only that application.

The claimed advantages of unikernels over traditional virtualization solutions generally include security and performance [17]. Security is improved by the reduced attack surface and increased isolation between applications; performance is improved by eliminating unnecessary components from the application.

While there are now a variety of unikernels suitable for general use, the objective advantages (or lack thereof) have not been explored in great detail. The creators of most of these systems have evaluated their performance in microbenchmarks which often show a performance advantage for a unikernel compared to a full operating system, but their real-world performance is generally left unexplored.

We present a set of realistic macrobenchmarks comparing the performance of several unikernel network applications (TCP/UDP, DNS, and HTTP) against a typical Linux deployment of the same applications. Our goal is to quantify the performance advantages of the unikernel approach to service deployment over traditional methods, and to guide prospective users to the best platform for their application. Our conclusions suggest that there is some significant merit in the unikernel method, but the unikernel implementations may not yet stable enough for widespread deployment in production.

## 2 Background

Most operating systems used today in virtual machines are mostly identical to the traditional, general-purpose operating systems used in non-virtualized environments. Since it is common today to configure virtual machines in the cloud to run a single application such as a web server or a database, a significant amount of the code, data, and services provided by the general-purpose operating system go unused and result in inefficiencies and wasted resources.

*Prepared for CS6480, Advanced Computer Networking, Fall 2014
†Project advisor

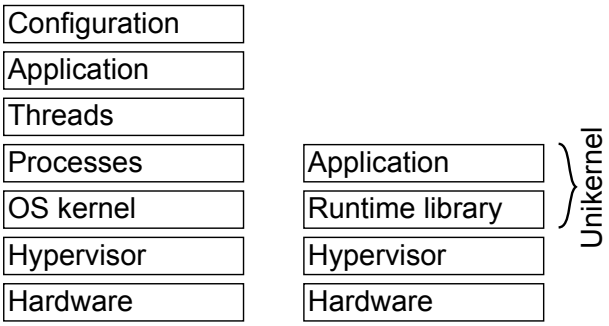


Figure 1: Traditional OS structure (left) vs unikernel

By eliminating unnecessary components, unikernel applications become much simpler and smaller, and thus are able to start up much more efficiently. For example, a web server unikernel can boot in less than a second. This method enables a new approach to the management of cloud resources: a virtual appliance, where a set of unikernels can be deployed cooperatively to build a distributed system. By rearchitecting traditional systems in this way, unikernel applications can, for example, respond to load spikes more elastically by spawning new unikernel VM instances without wasting significant resources. [17] The comparison between these architectures is visualized in Figure 1.

In some unikernels, such as Mirage OS, the cost of these benefits is paid at compile time, when static analysis is performed on the unikernel's code and configuration data to create the optimal boot image. Other unikernels, such as OSv, do not perform as much static analysis at compile time because they are designed to run programs in a more general-purpose way while still providing some benefits in flexibility and efficiency.

Existing unikernels are largely differentiated by the programming languages that they are implemented in, often taking advantage of safety guarantees enforced by the language itself. For example, Mirage OS [16], HaLVM [8], and Erlang on Xen [7] all provide language runtime systems with strong focus on type-safety (OCaml, Haskell, and Erlang, respectively), while ClickOS [18] provides a number of primitives meant for implementation of network middleboxes in C++. OSv [12], a relative newcomer, provides an optimized Java virtual machine in addition to its ability to run most POSIX programs with minimal modification.

# 3 Platforms Evaluated

We selected two popular unikernel platforms to evaluate: Mirage OS and OSv. These platforms are both relatively young; Mirage was first released in March 2013, and OSv was first released in September 2013. These platforms seemed to be sufficiently stable and mature that useful evaluation could be performed. Linux was chosen for its stability and popularlity to provide baseline data.

## 3.1 Mirage OS

We chose the latest stable version of Mirage OS, version 2.0 [19], for evaluation. Mirage only has support for the widely-used Xen hypervisor. Mirage is implemented entirely in the OCaml programming language; its authors cite [15] OCaml's brevity and static typing for reducing the attack surface of its code by eliminating common security exploits such as buffer and stack overruns. The authors also tout OCaml's high-performance runtime and compatibility with the Xen Cloud Platform and critical system components which are implemented in OCaml. Specifically, static analysis at compile-time enables Mirage apps to be streamlined.

## 3.2 OSv

OSv builds on large existing bodies of code, with much of its driver layer coming from BSD. The core of the system is implemented in C and C++, borrowing large amounts of the support infrastructure from existing POSIX systems. OSv supports the Linux ABI and most POSIX APIs in addition to allowing applications to use the OSv C++ API.

All threads in OSv share a single address space. In POSIX semantics, this means system calls such as `fork` are not available (because forking a process creates a new address space), but `clone` remains available for spawning new threads. The single address space approach permits significant performance improvements- context switching between threads does not incur a TLB flush, nor does performing a system call incur context switch overhead. The non-POSIX OSv APIs take additional advantage of this, such as by allowing zero-copy network transmit and receive operations.

OSv's implementation as a BSD-based core exposing Linux ABI has some unusual implications where applications attempt to make use of Linux-specific or otherwise non-standard interfaces. For example, our DNS server application (subsection 5.2) uses ancillary socket data to set the ToS bits on transmitted UDP datagrams when performing recursive lookups. Platform-specific values (such as the values of socket operation flags) must be translated between the Linux and BSD ABIs, and in the case of socket ancillary data this translation is not implemented. In that case, we modified the OSv syscall to strip ancillary data in calls to `send`, which had no ill effect on application functionality.

### 3.3 Linux

To obtain a useful set of baseline performance data for comparison, we also ran our tests against comparable services running on the Linux [14] operating system, itself running unmodified on the Xen hypervisor. This platform was based on Ubuntu [4] 14.04 for x86_64.

## 4 Service Performance Model

To process client requests, a service requires CPU time, free memory, and network bandwidth. For every request received, the server must perform some processing, during which CPU time and memory are consumed in addition to network bandwidth being consumed to return a response to the client.

In our benchmarks, we expect most workloads to be limited by CPU time. As the request rate rises, the server eventually reaches a point where there is insufficient CPU time to process requests immediately as they arrive, so some requests must be buffered while they are processed as quickly as possible. Buffering introduces additional latency between reception of a request and transmission of the corresponding response, so at this point we expect the response rate as measured at the client to stop increasing and response latency to increase.

Though Mirage provides introspective methods to determine the memory and CPU use, we opted not to use those methods to measure Mirage because we lack equivalents of those tools for all platforms and we believe these introspected measurements to be less reliable than external measurements. OSv exposes a count of free memory via the management API but does not provide CPU accounting data, while Linux has a wide variety of userspace accounting tools for both CPU and memory use but these tools are intended for measuring userspace applications rather than the system as a whole.

We opted not to measure memory use in benchmarks via the mechanisms available to us due to the potential impact on guest performance. The Linux guest in particular would need to run at least one more process to poll memory usage, and in all cases reporting memory use introduces another thread into the system which must compete for resources. External introspection is not a feasible method for memory measurement either, because Xen reserves all of the physical memory for a DomU at the time of creation, performing no additional accounting on guest page table allocations.

Though our hypervisor provides methods to perform CPU time accounting with the `xentop` tool, we have not captured CPU data for any of our benchmarks. Synchronizing capture of data between the client and server is a situation our evaluation tools do not currently handle well, suggesting room for future improvement. Our capture of client-side data only means we are unable to definitively identify the limiting factors in performance, though we are able to make educated guesses given our knowledge of the platforms.

## 5 Selected Applications

To evaluate the relative performance of each platform, we selected three different kinds of network services for evaluation: low-level (TCP/UDP), mid-level (DNS), and high-level (HTTP).

### 5.1 TCP/UDP

To measure the performance of the TCP and UDP network protocols, we selected the Iperf [11] benchmark. Our goal was to measure TCP bandwidth, UDP bandwidth, and UDP delay jitter.

We expect all platforms to easily saturate a gigabit network connection, though CPU utilization differences while doing so may indicate differences in the network stack's efficiency between the platforms. Similarly, variance in UDP delay jitter may indicate scheduler-based inefficiencies in individual platforms.

The Linux server for this test was unmodified Iperf version 2.0.5, and in all tests the client was also Iperf 2.0.5.

On OSv, the Iperf server was version 2.0.5 modified to perform zero-copy I/O with OSv-specific APIs. These changes total approximately 50 lines of code, and are included with the OSv source distribution as `iperf-zcopy` [5]. Due to the zero-copy I/O support, we expect it to be slightly more CPU efficient.

On Mirage, the Iperf server was an "Iperf-like" clone written by Dimosthenis Pediaditakis in OCaml for the Mirage platform [6].

### 5.2 DNS server

The DNS server for each test was configured as the authoritative name server for a fixed domain, and the benchmark client made requests for a single member of that domain at a target request rate for ten seconds. The latency of each request was measured, yielding a composite measurement of server response rate (in queries per second).

The Linux platform's DNS server was BIND [10] 9.8.4, as packaged by the distribution and reconfigured with our zone file only.

The OSv server was built from BIND 9.10.1 sources, ported to OSv. The BIND server did not require modification beyond minor patches to the build system to generate a shared object library rather than ELF binary and

corresponding build-time configuration to disable features that are incompatible with shared object generation (notably, the built-in capability to provide stack traces).

To support BIND, we needed to make several source-level changes to OSv. BIND refers to several C library functions and symbols that are not currently provided by OSv, so we stubbed out the functions that are not critical to BIND's functionality (notably `syslog`) and implemented the others in terms of existing functionality (`__strsep_g` and `sigsuspend`). We also modified the `sendmsg` syscall to ignore all ancillary data as described in subsection 3.2.

The Mirage platform's DNS server was the skeleton DNS server [20] provided by the Mirage team. This example DNS server does not support most of the features provided by the BIND server; indeed, the only feature that it supports is responding to UDP DNS queries for one particular hard-coded zone. Some minor customization of the Mirage DNS server was required: in addition to configuring the DNS zone and the server address and port for our environment, we modified the DNS server code so that it did not output a log message for every received request. Prior to this change, the performance was very poor.

### 5.3 HTTP server

Performance of an HTTP server serving static content was measured by generating HTTP requests for a single static resource (a 136-byte HTML file) at varying rates and measuring the response time for each request. This yields a composite measurement of server response rate.

As a proxy for introspected memory use measurements (discussed in Section 4), we reduced the DomU memory size by a factor of two from the original 1 gigabytefor each subsequent test until the server failed or the response rate was reduced to less than 25 percent of its baseline value. This approach allows us to infer the memory requirements for the application without directly measuring memory use.

To test HTTP service on Linux, we chose Apache HTTP Server [1] version 2.4.7 as packaged for Ubuntu. No changes were made to the default Apache server configuration.

OSv includes a web server for its management application, but this server is not well-suited to adaptation for other uses. The source distribution includes application servers as well, but these are not suitable for benchmarking a static web site. We instead ported lighttpd [13], which required no patches other than configuring it to build the server binary as a shared library object. We found that there appears to be a serious bug in OSv which made it impossible to test lighttpd on OSv with Xen,[1] so we instead ran the OSv lighttpd image under Linux's KVM hypervisor, where the bug does not appear to manifest.

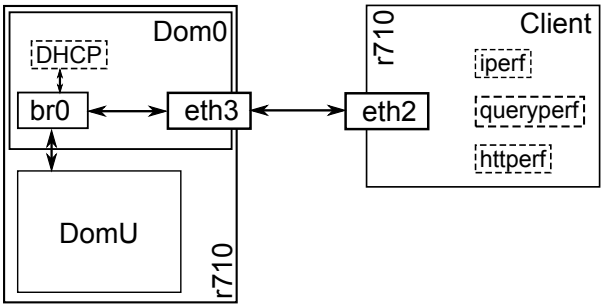


Figure 2: Experiment configuration in testbed

The Mirage platform's HTTP server was the skeleton HTTP server [21] provided by the Mirage team. This example HTTP server does not support most of the features provided by the other HTTP servers; for example, it does not support plugin modules for enabling third-party features like SSL or PHP.

## 6 Evaluation

### 6.1 Experiment setup

All platforms were evaluated running in a Xen 4.4 guest domain on Dell r710 servers with Intel Xeon E5530 processors and 12 gigabytes of RAM in the Emulab [24] network testbed. Benchmark client programs were run on identical r710 servers connected via a dedicated gigabit Ethernet link. Both client and Dom0 operating systems were Ubuntu 14.04. This configuration is shown in Figure 2.

Guest domains were allocated one gigabyte of memory and one virtual CPU, with a virtual network interface attached to a Linux network bridge device on the host, which was in turn connected to the dedicated client-server link. The Xen Dom0 and client machine were assigned static IP addresses on this private network, while Xen DomU guests were assigned IP addresses by a DHCP server running in the Xen Dom0.

As test clients, we used the following three applications:

- Iperf [11], a bandwidth measurement tool
- queryperf [22], a DNS response measurement tool from the ISC BIND "contrib" distribution
- httperf [9], a tool for web server performance testing from HP Labs

## 6.2 Iperf

Unfortunately, the only available Mirage Iperf server was written for Mirage OS version 1.0, which is not compatible with the version 2.0 that we tested. As a result, we were unable to obtain Iperf data for Mirage. We tested OSv and Linux just enough to confirm that Iperf can easily saturate the network link, as we expected.

## 6.3 DNS

The results of the three DNS benchmarks are shown in Figure 3, Figure 4, and Figure 5. Note that the Linux and OSv graphs have an identical X axis, while the request rate for Mirage goes up to 80 thousand.

The DNS server running on Linux hits its limit at about 19,000 requests/second, at which point response latencies become significantly higher.

The OSv DNS server performs significantly better than Linux both in terms of response latency and maximum request rate. It runs out of steam at about 30,000 requests/second while consistently responding faster to requests.

The Mirage DNS server is able to sustain a higher request rate than Linux or OSv, but at some cost: a small fraction of the response times are significantly higher than the average. These outliers, indicated by the gray X's on the chart, are much more abundant at higher latencies than on Linux or OSv. We speculate that this is due to memory garbage collection being performed occasionally by the OCaml unikernel runtime. The Mirage OS authors also mentioned in their 2013 paper that memoization of DNS responses roughly doubled performance; perhaps this would help as well.[2]

## 6.4 HTTP

The results of our HTTP benchmarks for all three platforms are shown in Figure 6. Shaded regions on the graph represent an increase of standard deviation from the average latency for each given request rate. `httperf` does not export fine-grained statistics in the same way `queryperf` does, so we were unable to perform additional statistical processing on the data as was done with DNS results.

The Linux HTTP server performs well until about 4,000 requests/second, at which point it becomes unable to keep up with the requests and its performance drops off significantly.

The OSv HTTP server performs consistently well all the way through 5,000 requests/second. We speculate that the increase in response latency is due to client resource exchaustion, described below.

We were unable to test higher than about 5,000 requests/second using a single test client because httperf creates a new TCP connection for each HTTP request, in order to simulate real-world requests coming from different browsers, and a critical TCP resource becomes almost entirely consumed at that rate: each time a TCP connection is closed, it must remain in the TCP TIME_WAIT state for about one minute, by default, on Linux. So, after creating 5,000 requests/second for ten seconds, there are 50,000 TCP connections on the Linux client, each tying up a unique source TCP port. So, in order to test higher HTTP request rates, we would have to redesign our test to support multiple clients, or change the default TCP TIME_WAIT timeout, and we ran out of time to explore these options. As a result, we do not yet know the upper limit of OSv HTTP server's response rate, but it is appears to perform better than Linux.

During testing of the Mirage HTTP server we encountered a severe bug: the Mirage TCP/IP library apparently leaks memory every time a TCP connection is released, on the order of 30 kilobytes per TCP connection. We have filed a bug with the Mirage OS authors. Unfortunately, this causes the performance of Mirage to be very poor; we expect this performance would be greatly improved after this bug is fixed.

OSv HTTP didn't see any performance degradation when its total memory allocation in Xen was reduced to 256 MB. At 128 MB of memory, OSv performed normally up to 4,000 requests per second, at which point the server crashed due to out-of-memory conditions. Linux's performance was not affected by memory size either, and it maintained the same level of performance down to 256 MB of memory, which is the stated minimum requirement for Ubuntu Server.

# 7 Future Work

We have identified several further areas that we would like to explore in the future, including:

- Fixing the TCP/IP stack bug that caused Mirage HTTP to perform so poorly. (See Figure 6.4.)
- Fixing the bug that prevented running lighttpd on OSv on Xen. (See subsection 5.2.)
- Testing fully-featured Mirage network servers, when they become available, instead of the example skeleton servers that are available today. For example, a Mirage HTTP server that has reasonable feature parity with the Apache HTTP Server.
- Testing the performance of a dynamic web server application; for example, a web page containing content that must be retrieved from a database.

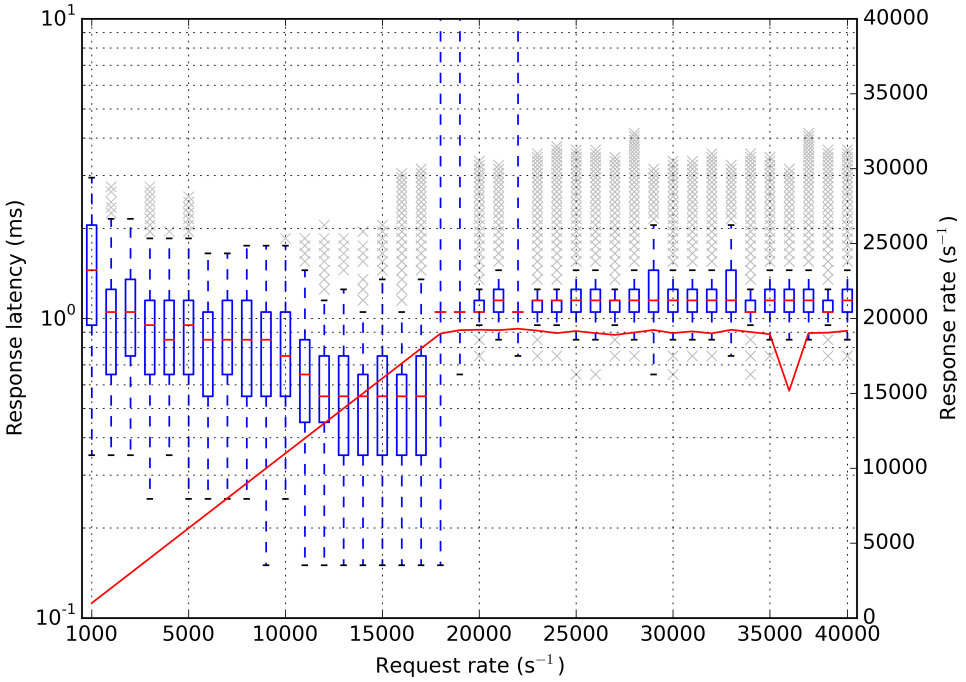


Figure 3: Linux DNS server results.

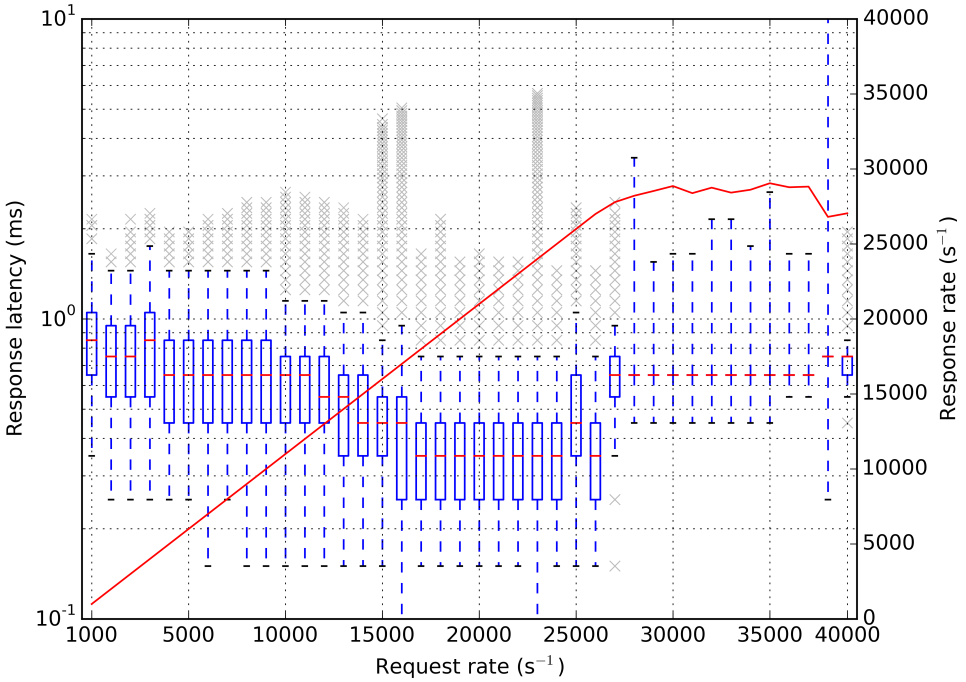


Figure 4: OSv DNS server results.

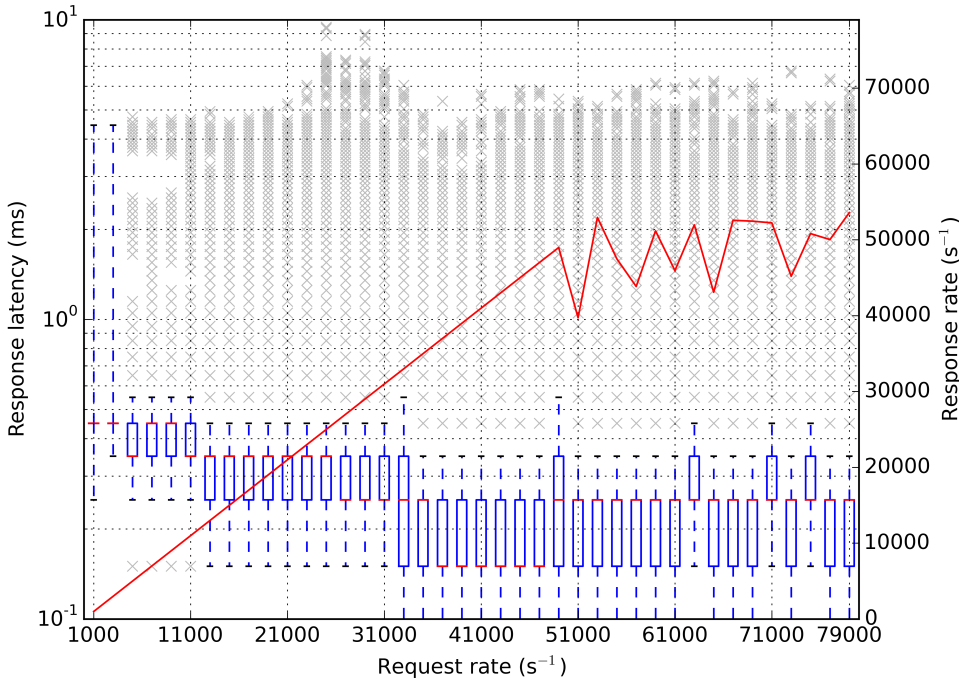


Figure 5: Mirage DNS server results.

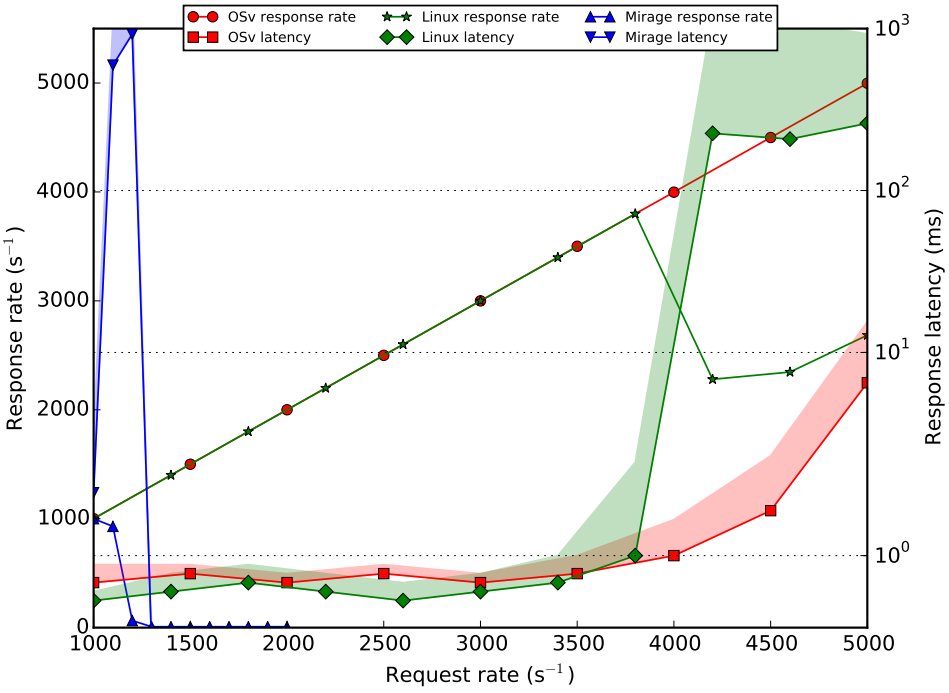


Figure 6: HTTP Server Benchmark Results

- Testing the performance of recursive DNS server resolution, since that would be more representative of a real-world environment.
- Testing the performance of operating system-level virtualization such as the Linux control group (`cgroups`) feature as used by Docker and LXC.
- Improving our test rig so that we can test the establishment of greater than 5,000 TCP connections/second. (See Figure 6.4.)
- Testing other unikernel platforms, such as Erlang on Xen and HaLVM.
- Measuring more unikernel characteristics; for example, boot time, memory requirements, and reliability.

## 8 Conclusion

The two unikernel platforms we evaluated are both so different and so young that it is unreasonable to draw any general conclusions about the unikernel architectural model. However, the results we were able to obtain are promising.

OSv significantly exceeded the performance of Linux in every category, while not requiring much work in order to port the chosen applications. But, all was not perfect with OSv; we were not able to run lighttpd on OSv on Xen, due to a crashing bug. Given that OSv is currently considered alpha software by its authors, these results suggest production versions of OSv will be very attractive for high-performance applications though its current stability and dearth of well-tested applications is not appropriate for production deployment.

Mirage OS's performance is more of a mixed bag than the other two platforms. Neither the DNS server nor the HTTP server are ready for deployment in production because they were merely example skeleton servers that contain low-level debug print statements and are missing key features. Furthermore, a serious bug in its TCP module prevented us from evaluating its HTTP server. However, the DNS server was able to sustain a request rate that was significantly higher than both Linux and OSv. This suggests that it may be worth trying Mirage again after it has had more time to mature and stabilize.

It is noteworthy that OSv supports existing POSIX applications without much modification, while Mirage only supports OCaml applications that have been specifically ported to the Mirage framework. This gives OSv the great advantage of compatibility with a huge number of existing applications, but it lacks the security and performance benefits offered by OCaml's type-safety and static analysis at compile-time.

Though we found the stability and feature sets of these unikernels to be somewhat lacking when compared aginst the Linux testbed, the performance data we were able to capture are promising for the prospects of future deployment in real-world applications, providing good security isolation with smaller resource footprint than monolothic "traditional OS" virtual machines and comparable (if not better) performance.

## References


[1] Apache Team. Apache http server project. http://httpd.apache.org/.

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM 53*, 4 (Apr. 2010), 50–58.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 164–177.

[4] Canonical Ltd. Ubuntu operating system. http://www.ubuntu.com/.

[5] Cloudius Systems. Zero-copy iperf for osv. https://github.com/cloudius-systems/osv-apps/tree/master/iperf-zcopy.

[6] Dimosthenis Pediaditakis. Iperf-like utility for mirage os. https://github.com/dimosped/iperf-mirage.

[7] Erlang on Xen Team. Erlang on xen. http://erlangonxen.org/.

[8] Galois, Inc. Haskell lightweight virtual machine (halvm). https://galois.com/project/halvm/.

[9] Hewlett-Packard Research Laboratories. httperf tool for measuring web server performance. http://www.hpl.hp.com/research/linux/httperf/.

[10] Internet Systems Consortium. Bind dns software. http://www.isc.org/downloads/bind/.

[11] Iperf Team. Iperf. https://iperf.fr/.

[12] Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D., and Zolotarov, V. Osv: Optimizing the operating system for virtual machines. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 61–72.

[13] lighttpd Team. Apache http server project. http://www.lighttpd.net/.

[14] Linux Team. The linux kernel. https://www.kernel.org/.

[15] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., and Crowcroft, J. Unikernels: Library operating systems for the cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2013), ASPLOS '13, ACM, pp. 461–472.

[16] Madhavapeddy, A., Mortier, R., Sohan, R., Gazagnaire, T., Hand, S., Deegan, T., McAuley, D., and Crowcroft, J. Turning down the lamp: Software specialisation for the cloud. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 11–11.

[17] MADHAVAPEDDY, A., AND SCOTT, D. J. Unikernels: Rise of the virtual library operating system. *Queue 11*, 11 (Dec. 2013), 30:30–30:44.

[18] MARTINS, J., AHMED, M., RAICIU, C., AND HUICI, F. Enabling fast, dynamic network processing with clickos. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 67–72.

[19] MIRAGE OS TEAM. Mirage os. http://www.openmirage.org/.

[20] MIRAGE TEAM. Skeleton dns server for mirage os. https://github.com/mirage/mirage-skeleton/tree/master/dns.

[21] MIRAGE TEAM. Skeleton http server for mirage os. https://github.com/mirage/mirage-skeleton/tree/master/static_website.

[22] NOMINUM, INC. queryperf dns query performance testing tool. http://ftp.isc.org/isc/bind9/9.9.0rc1/bind-9.9.0rc1/contrib/queryperf/.

[23] USENIX. Usenix atc '15 call for papers. https://www.usenix.org/conference/atc15/call-for-papers.

[24] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.

## Notes

[1] The breaking bug in OSv appears to be threading-related, in which the server properly responds to the first request made to it after starting up but following which the system becomes completely unresponsive to input both on the network and its local console.

[2] Unfortunately, this memoization code was accidentally deleted by the Mirage OS authors after publication of their 2013 paper, so we were unable to test it.