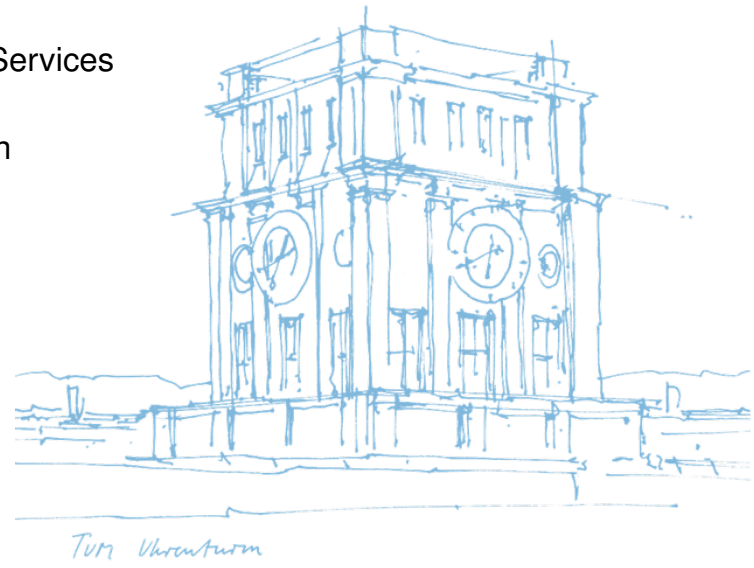
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Networking in MirageOS

Fabian Bonk
advised by Paul Emmerich

Friday 18th January, 2019

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich



What is MirageOS?

MirageOS is a library operating system that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

Unikernels

What's a Unikernel?

Unikernels

What's a Unikernel?

- Entire application compiled into bootable VM image

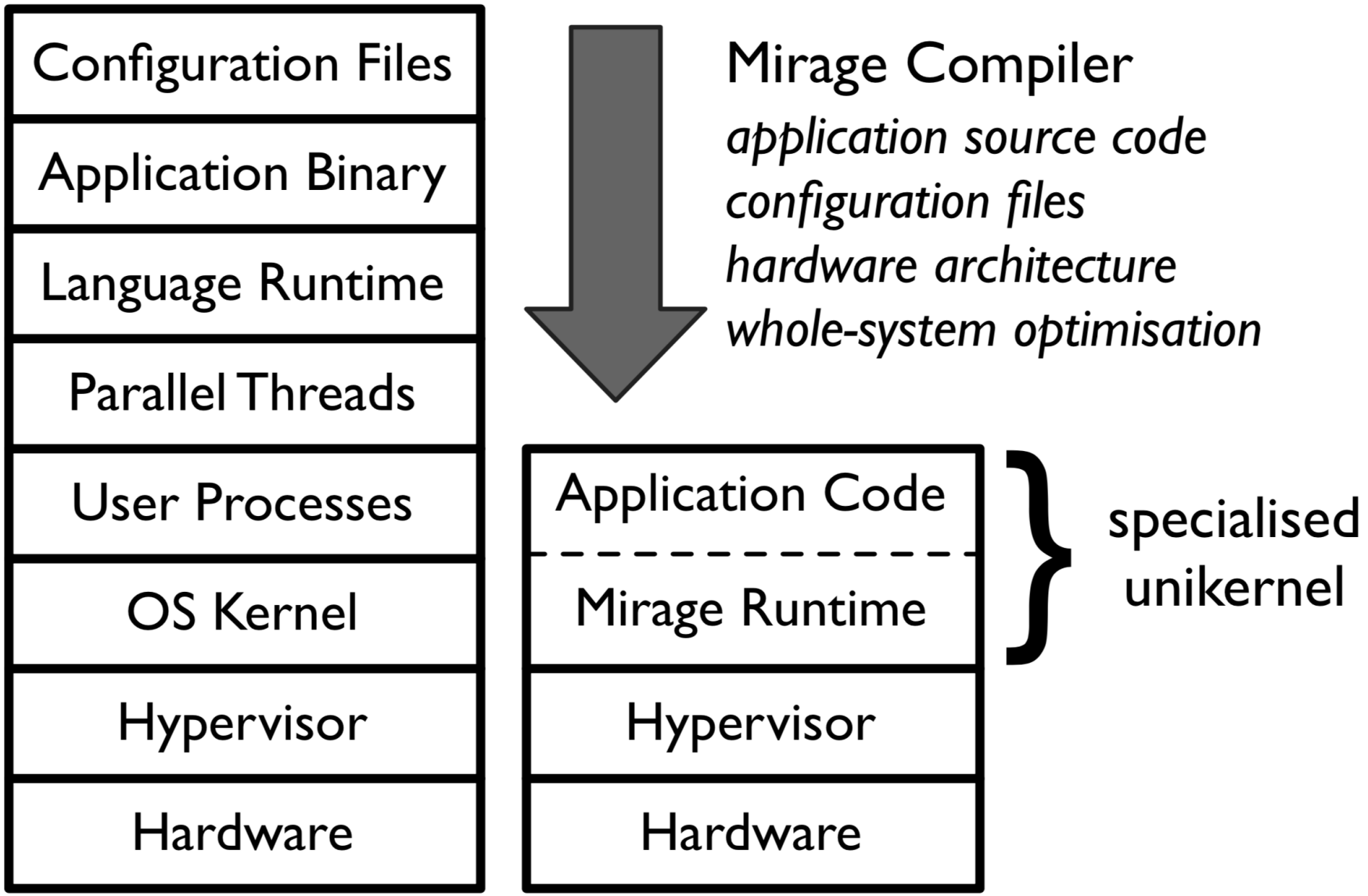
Unikernels

What's a Unikernel?

- Entire application compiled into bootable VM image
- Include necessary operating system functionality via libraries

Unikernels

Unikernels vs. VMs



[1]

Unikernels

Why Unikernels?

- high degree of separation
- low resource usage
- flexible runtime(s) (run on hypervisors, standard OS, microcontrollers)
- safety benefits of high-level languages
- fewer loc $\rightarrow$ fewer bugs

MIRAGE OS

A Cloud Operating System

MirageOS

OCaml unikernel operating system

`https://mirage.io/`

- written in OCaml
- generates Xen (incl. QubesOS) and Solo5 (KVM) Unikernels
- can also generate standard executables (Linux, macOS, ...)
- 153 + 11 repos

OCaml

OCaml

MirageOS

OCaml

OCaml is

- functional
- object-oriented (optional)
- imperative (optional)
- compiled (machine code + bytecode)
- statically typed (with type-inference)
- garbage-collected
- single-threaded :(

MirageOS

Network protocols

- Ethernet and ARP
- IPv4 and IPv6
- ICMP
- TCP
- UDP
- TLS
- HTTP
- DNS
- DHCP

MirageOS

Other libraries

- nocrypto (AES, RSA, DH, SHA, HMAC)
- ocaml-git
- ocaml-clock
- decompress (zlib)
- ocaml-pcap
- mirage-block-ramdisk
- ocaml-tar
- metrics
- ocaml-cstruct

MirageOS

Example: Echo server

```
open Lwt.Infix

module Main (S : Mirage_types_lwt.STACKV4) = struct
  (* RFC 862 - read payloads and repeat them back *)
  let rec echo flow =
    S.TCPV4.read flow >>= function
    | Error e
    | Ok `Eof -> S.TCPV4.close flow
    | Ok (`Data buf) ->
      S.TCPV4.write flow buf >>= function
      | Error e -> S.TCPV4.close flow
      | Ok () -> echo flow

  let start s =
    S.listen_tcpv4 s ~port:7 echo;
    S.listen s
end
```

MirageOS

Example: Echo Server

How to build:

MirageOS

Example: Echo Server

How to build:

Build a normal binary and use a TAP device and the OCaml network stack:

```
$ mirage configure -t unix --net direct && make
```

MirageOS

Example: Echo Server

How to build:

Build a normal binary and use a TAP device and the OCaml network stack:

```
$ mirage configure -t unix --net direct && make
```

Build a normal binary and use the OS network stack:

```
$ mirage configure -t unix --net socket && make
```

MirageOS

Example: Echo Server

How to build:

Build a normal binary and use a TAP device and the OCaml network stack:

```
$ mirage configure -t unix --net direct && make
```

Build a normal binary and use the OS network stack:

```
$ mirage configure -t unix --net socket && make
```

Build a standalone Unikernel for deployment on Xen:

```
$ mirage configure -t xen && make
```

Cstruct

What is it?

- library for accessing raw memory (read, write, endianness swapping, etc.)
- new type `Cstruct.t` that wraps raw memory

Cstruct

ppx_cstruct

OCaml preprocessor for autogenerating accessors

Cstruct

ppx_cstruct example

```
[%%cstruct
  type udp_header = {
    sport : uint16;
    dport : uint16;
    length : uint16;
    checksum : uint16
  } [@@big_endian]
]
```

Cstruct

ppx_cstruct example

```
[%%cstruct
  type udp_header = {
    sport : uint16;
    dport : uint16;
    length : uint16;
    checksum : uint16
  } [@@big_endian]
]
```

```
val sizeof_udp_header : int
val get_udp_header_sport :
  Cstruct.t -> int
val set_udp_header_sport :
  Cstruct.t -> int -> unit
val get_udp_header_dport :
  Cstruct.t -> int
val set_udp_header_dport :
  Cstruct.t -> int -> unit
val get_udp_header_length :
  Cstruct.t -> int
val set_udp_header_length :
  Cstruct.t -> int -> unit
val get_udp_header_checksum :
  Cstruct.t -> int
val set_udp_header_checksum :
  Cstruct.t -> int -> unit
val hexdump_udp_header_to_buffer :
  Buffer.t -> Cstruct.t -> unit
val hexdump_udp_header :
  Cstruct.t -> unit
```

Cstruct

ppx_cstruct example

```
[%%cenum
  type ethertype =
   | IPv4 [@id 0x0800]
   | IPv6 [@id 0x86DD]
   | ARP [@id 0x0806]
   [@@uint16]
]
```

Cstruct

ppx_cstruct example

```
[%%cenum
  type ethertype =
   | IPv4 [@id 0x0800]
   | IPv6 [@id 0x86DD]
   | ARP [@id 0x0806]
   [@@uint16]
]
```

```
type ethertype = IPv4 | IPv6 | ARP
val int_to_ethertype :
  int -> ethertype option
val ethertype_to_int :
  ethertype -> int
val ethertype_to_string :
  ethertype -> string
val string_to_ethertype :
  string -> ethertype option
```

Mirage Network Interfaces

Abstraction

- Network interfaces are behind a module signature (aka interface)
- easily implemented

Mirage Network Interfaces

Mirage_net.S

```
module type Mirage_net.S =
  sig
    type error = private [> Mirage_device.error ]
    val pp_error : error Fmt.t
    type page_aligned_buffer
    type buffer
    type macaddr
    type +'a io
    type t
    val disconnect : t -> unit io
    val write : t -> buffer -> (unit, error) result io
    val writev : t -> buffer list -> (unit, error) result io
    val listen : t -> (buffer -> unit io) -> (unit, error) result io
    val mac : t -> macaddr
    val get_stats_counters : t -> Mirage_net.stats
    val reset_stats_counters : t -> unit
  end
```

Mirage Network Interfaces

Mirage_net.S

```
module type Mirage_net.S =
  sig
    (* ... *)
    val disconnect : t -> unit io
    val write : t -> buffer -> (unit, error) result io
    val writev : t -> buffer list -> (unit, error) result io
    val listen : t -> (buffer -> unit io) -> (unit, error) result io
    val mac : t -> macaddr
    val get_stats_counters : t -> Mirage_net.stats
    val reset_stats_counters : t -> unit
  end
```

Mirage Network Interfaces

Implementations

Multiple implementations:

- mirage-net-unix
- mirage-net-xen
- mirage-net-macosx
- mirage-net-flow
- mirage-net-fd
- mirage-net-solo5

Mirage Network Interfaces

Implementations

Multiple implementations:

- mirage-net-unix
- mirage-net-xen
- mirage-net-macosx
- mirage-net-flow
- mirage-net-fd
- mirage-net-solo5

Xen

Network devices

- network device abstractions provided by hypervisor
- wait for event signal from Xen
- copy packet from shared memory onto OCaml heap
- reverse for tx

KVM

Solo5

`https://github.com/solo5/solo5`

- runtime environment for unikernels
- supports Linux, FreeBSD and OpenBSD
- network access via virtio (deprecated) or TAP

KVM

mirage-solo5

- Small C library that wraps hypercalls as OCaml functions

KVM

mirage-net-solo5

- OCaml library that calls into mirage-solo5
- passes pointers to packet buffers to the Solo5 host

ixy.ml

OCaml userspace network driver

- userspace network driver for Intel 82599 NICs written in OCaml
- cstructs for packet buffers
- packet buffers allocated in Linux hugepages
- manual allocation/deallocation of packet buffers
- support for batch rx/tx
- >7 Mpps bidirectional forwarding

Performance

OCaml lists

- singly-linked
- immutable
- iterated over using recursion
- frequent reversing

Performance

cstructs

- full copy for sending and receiving required
- frequent allocations and more copies during packet assembly
- easy garbage collection

Performance

Batching

- no batching
- individual handling of every packet

Performance

parallelism

- OCaml has no multicore support
- concurrency via Lwt (promises)
- ocaml-multicore soon™

Examples

Example: MirageOS website

`https://mirage.io/`

OCaml all the way:

- network stack
- filesystem
- webserver (HTTP + HTTPS)
- client-side statistics: OCaml compiled to JavaScript
- around 2700 loc (without HTML, JS, CSS, deployment scripts)

Examples

Example: QubesOS firewall

`https://github.com/talex5/qubes-mirage-firewall`

- uses 30 MB of memory (vs. up to 1GB for a Linux VM)
- about 800 loc
- change rules $\rightarrow$ recompile firewall

Examples

Example: Bitcoin Pinata

`http://ownme.ipredator.se/`

`https://mirage.io/blog/bitcoin-pinata-results`

- TLS server that knows the private key to a BTC address and is happy to send it to anybody

Examples

Example: Bitcoin Pinata

`http://ownme.ipredator.se/`

`https://mirage.io/blog/bitcoin-pinata-results`

- TLS server that knows the private key to a BTC address and is happy to send it to anybody
- connect, receive private key, walk away with 10 BTC?

Examples

Example: Bitcoin Pinata

`http://ownme.ipredator.se/`

`https://mirage.io/blog/bitcoin-pinata-results`

- TLS server that knows the private key to a BTC address and is happy to send it to anybody
- connect, receive private key, walk away with 10 BTC?
- twist: only after TLS client auth with cert signed by public CA cert

Examples

Example: Bitcoin Pinata

`http://ownme.ipredator.se/`

`https://mirage.io/blog/bitcoin-pinata-results`

- TLS server that knows the private key to a BTC address and is happy to send it to anybody
- connect, receive private key, walk away with 10 BTC?
- twist: only after TLS client auth with cert signed by public CA cert
- 200k attempts, 50k unique IPs

Examples

Example: Bitcoin Pinata

`http://ownme.ipredator.se/`

`https://mirage.io/blog/bitcoin-pinata-results`

- TLS server that knows the private key to a BTC address and is happy to send it to anybody
- connect, receive private key, walk away with 10 BTC?
- twist: only after TLS client auth with cert signed by public CA cert
- 200k attempts, 50k unique IPs
- $\rightarrow$ never claimed

Bibliography

A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, “Unikernels: Library Operating Systems for the Cloud,” SIGPLAN Notices, vol. 48, pp. 461-472, March 2013