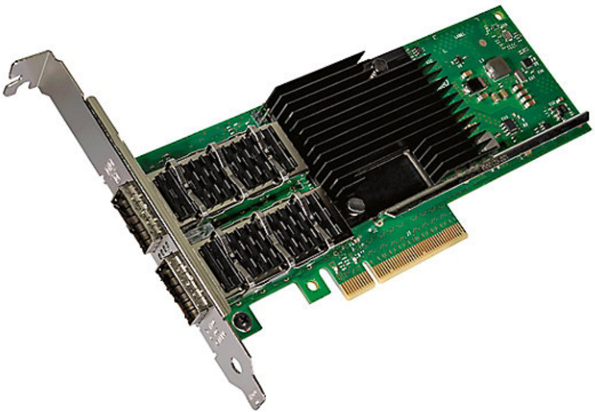
Safe and Secure User Space Drivers

Paul Emmerich, Simon Ellmann, Georg Carle

February 28, 2019

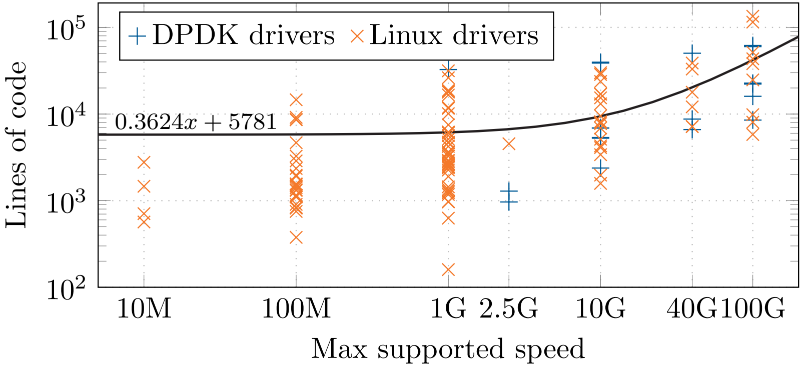
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Network drivers



Intel XL710 [Picture: Intel.com]

Network driver complexity is increasing

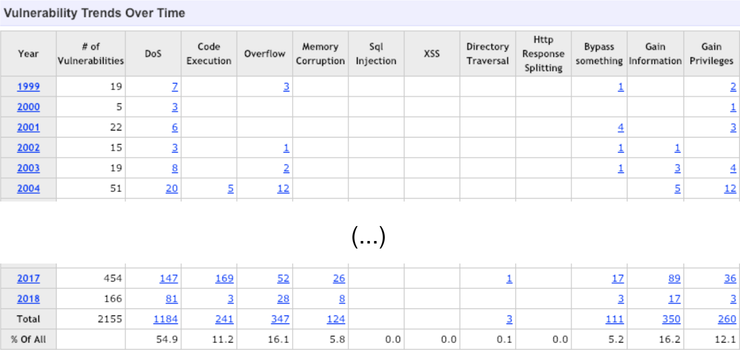


The ixy driver

- Our attempt to write a simple yet fast user space network driver
- It’s a user space driver you can easily understand and read
- Supports Intel ixgbe NICs (82599, X540, Xeon D, ...) and VirtIO
- $\approx$ 1,000 lines of C code, full of references to datasheets and specs
- Intel driver: 38,000 lines in DPDK, 30,000 in Linux
- Small code size makes it ideal for trustworthy systems

- But is C the best language for drivers?

C can cause security problems



Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges
1999	19	7		3						1		2
2000	5	3										1
2001	22	6								4		3
2002	15	3		1						1	1	
2003	19	8		2						1	3	4
2004	51	20	5	12							5	12
(...)												
2017	454	147	169	52	26			1		17	89	36
2018	166	81	3	28	8					3	17	3
Total	2155	1184	241	347	124			3		111	350	260
% Of All		54.9	11.2	16.1	5.8	0.0	0.0	0.1	0.0	5.2	16.2	12.1

- Screenshot from `https://www.cvedetails.com/`
- Security bugs found in the Linux kernel in the last ≈ 20 years

C can cause security problems

- Not all bugs can be blamed on the language
- Cutler et al. analyzed 65 CVEs categorized as code execution in the Linux kernel [1]

[1] C. Cutler, M. F. Kaashoek, and R. T. Morris, "The benefits and costs of writing a POSIX kernel in a high-level language", USENIX OSDI, 2018

C can cause security problems

- Not all bugs can be blamed on the language
- Cutler et al. analyzed 65 CVEs categorized as code execution in the Linux kernel [1]

Bug type	Num.	Perc.	Can be avoided by a high-level language?
Various	11	17%	Unclear/Maybe
Logic	14	22%	No
Use-after-free	8	12%	Yes
Out of bounds	32	49%	Yes (likely leads to panic)

Table 1: Code execution vulnerabilities in the Linux kernel identified by Cutler et al[1]

[1] C. Cutler, M. F. Kaashoek, and R. T. Morris, "The benefits and costs of writing a POSIX kernel in a high-level language", USENIX OSDI, 2018

Are there preventable bugs in drivers?

- We looked at these 40 preventable bugs

Are there preventable bugs in drivers?

- We looked at these 40 preventable bugs
- 39 of them were in drivers (the other was in the Bluetooth stack)

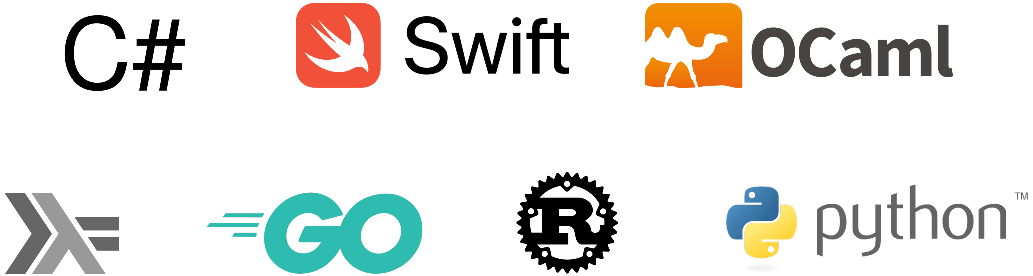
Should drivers for trustworthy systems be written in C?

- If you have a choice: probably not

Should drivers for trustworthy systems be written in C?

- If you have a choice: probably not
- User space drivers can be written in any language!
- But are all languages an equally good choice?
- Is a JIT compiler or a garbage collector a problem in a driver?

We wrote full user space drivers in these languages



Goals for our implementations

- Implement the same feature set as our C reference driver
- Use a similar structure like the C driver
- Write idiomatic code for the selected language
- Use language safety features where possible
- Quantify trade-offs for performance vs. safety

- This allows us to compare different languages for safety-critical systems

Language comparison: Overview

Language	Main paradigm	Memory management	Compilation
Rust	Imperative	Ownership/RAII	(LLVM) Compiled
Go	Imperative	Garbage collection	Compiled
C#	Object-oriented	Garbage collection	JIT
Swift	Protocol-oriented	Reference counting	(LLVM) Compiled
OCaml	Functional	Garbage collection	Compiled
Haskell	Functional	Garbage collection	(LLVM) Compiled
Python	Imperative	Garbage collection	Interpreted

Table 2: Language overview

Language comparison: Safety properties

	General memory		Packet buffers		
Language	Bounds checks	Use after free	Bounds checks	Use after free	Int overflows
C	✗	✗	✗	✗	✗
Rust	✓	✓	(✓)[1]	✓	(✓)[4]
Go	✓	✓	(✓)[1]	(✓)[3]	✗
C#	✓	✓	(✓)[1]	(✓)[3]	✗
Swift	✓	✓	✗[2]	(✓)[3]	✓
Haskell	✓	✓	(✓)[1]	(✓)[3]	✗
OCaml	✓	✓	(✓)[1]	(✓)[3]	✗
Python	✓	✓	(✓)[1]	(✓)[3]	✗

[1] Bounds enforced by wrapper, constructor in unsafe code

[2] Bounds only enforced in debug mode

[3] Buffers are never free'd, only returned to a memory pool

[4] Disabled by default, proposed to be enabled by default in the future

Table 3: Language-level protections against classes of bugs in our drivers

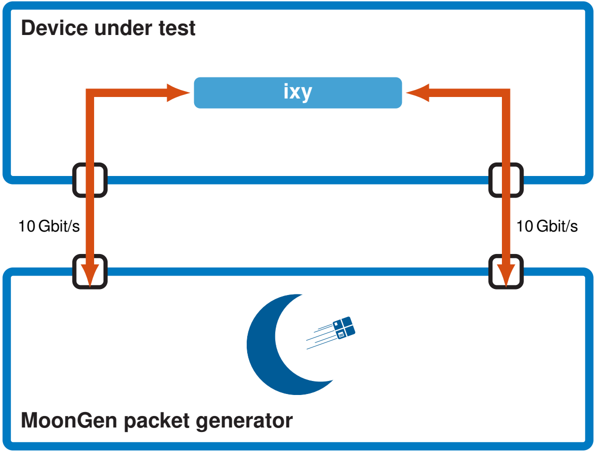
Language comparison: Implementation sizes

Lang.	Lines of code[1]	Lines of C code[1]	Code size (gzip[2])
C	831	831	12.9 kB
Rust	961	0	10.4 kB
Go	1640	0	20.6 kB
C#	1266	34	13.1 kB
Swift	1506	0	15.9 kB
Haskell	1001	0	9.6 kB
OCaml	1177	28	12.3 kB
Python	1242	(Cython) 77	14.2 kB

[1] Excluding empty lines and comments, counted with `cloc`
[2] Compression level 6

Table 4: Size of our implementations (w/o register offset constants, stripped features not found in all drivers)

Performance comparison: Test setup



Device under test

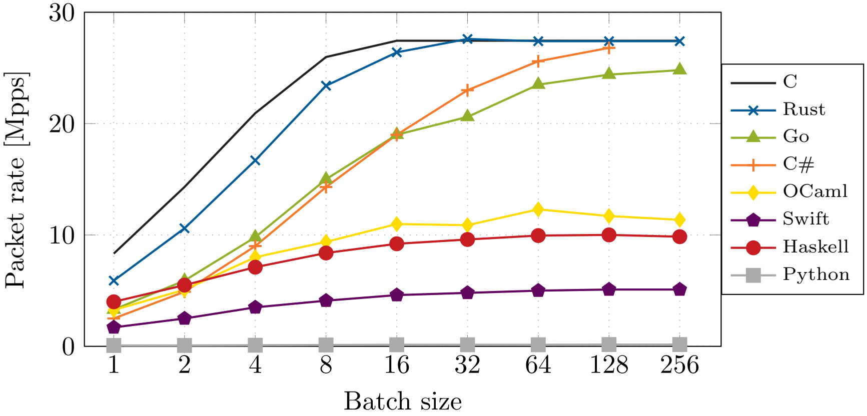
ixy

10 Gbit/s

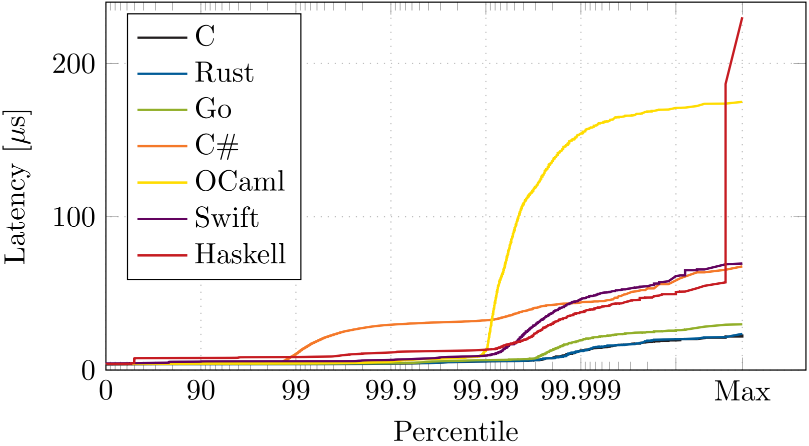
10 Gbit/s

MoonGen packet generator

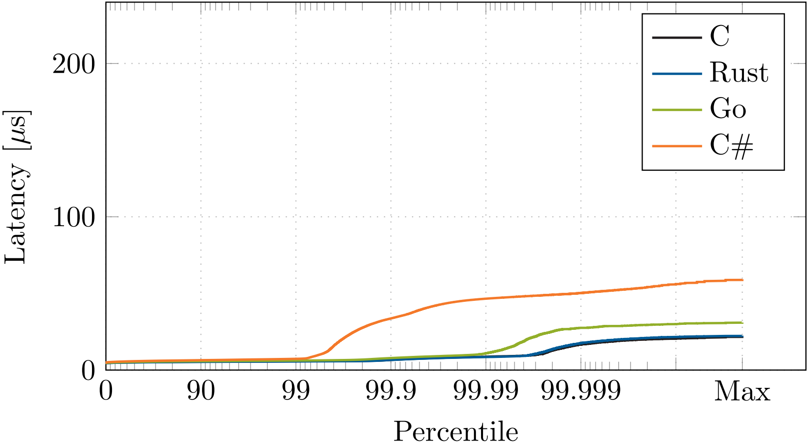
Batching at 3.3 GHz CPU speed



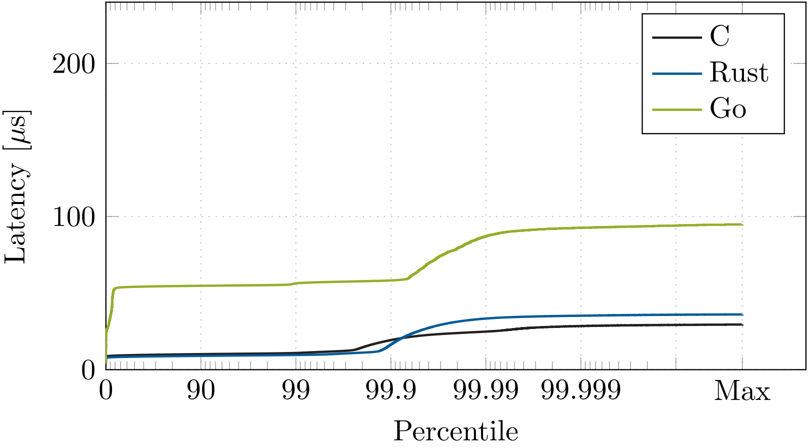
Tail latency at 1 Mpps



Tail latency at 10 Mpps



Tail latency at 20 Mpps



Languages for code in trustworthy systems

- Rust
  - Fast, no garbage collector
  - Low-level: Easy to reason about performance
  - Safest language of the evaluated languages
- Go
  - Fast, low-latency garbage collector
  - Garbage collector tuned for sub-millisecond latency
  - Easier and faster to write than Rust

Languages for code in trustworthy systems

- Rust
  - Fast, no garbage collector
  - Low-level: Easy to reason about performance
  - Safest language of the evaluated languages
- Go
  - Fast, low-latency garbage collector
  - Garbage collector tuned for sub-millisecond latency
  - Easier and faster to write than Rust
- Other languages
  - Implement critical parts in different languages in redundant systems
  - Functional languages for easier formal verification

Conclusions

- High-level languages can prevent entire classes of bugs
- High-level languages are suitable for low-level code
- Drivers are becoming more and more complex, simpler drivers reduce attack surface
- Future work: A full stack in Rust (ixy + smoltcp), evaluating Redox

- Paper about safer drivers under submission to SIGCOMM
- Code for all drivers available on GitHub:
  `https://github.com/ixy-languages/ixy-languages`

Backup: Unprivileged user space drivers

- User space drivers usually run with root privileges, but why?

Backup: Unprivileged user space drivers

- User space drivers usually run with root privileges, but why?

- Mapping PCIe resources requires root
- Allocating non-transparent huge pages requires root
- Locking memory requires root

- Can we do that in a small separate program that is easy to audit and then drop privileges?

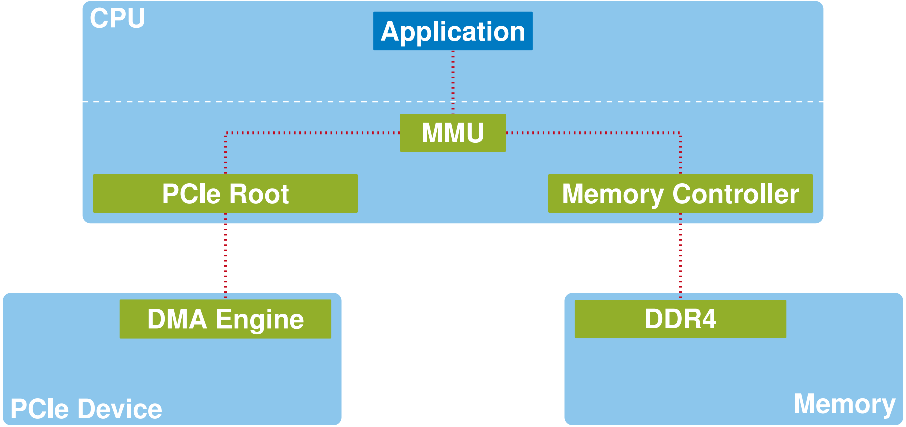
Backup: Unprivileged user space drivers

- User space drivers usually run with root privileges, but why?

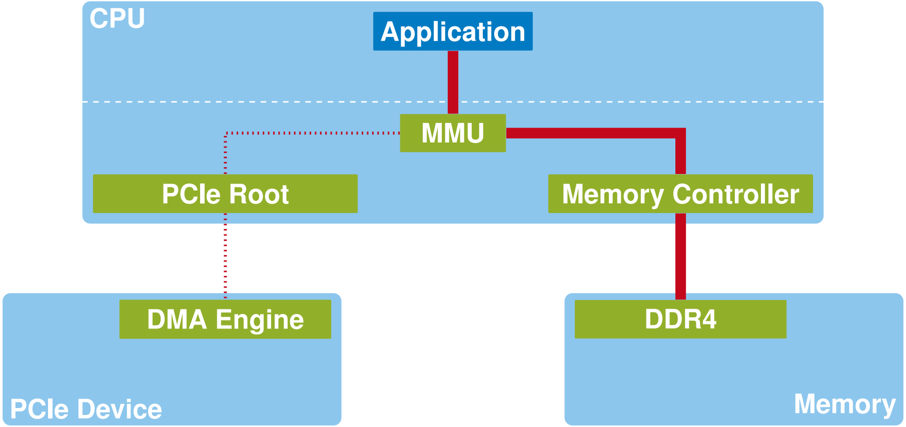
- Mapping PCIe resources requires root
- Allocating non-transparent huge pages requires root
- Locking memory requires root

- Can we do that in a small separate program that is easy to audit and then drop privileges?
- Yes, we can
- But it's not really secure

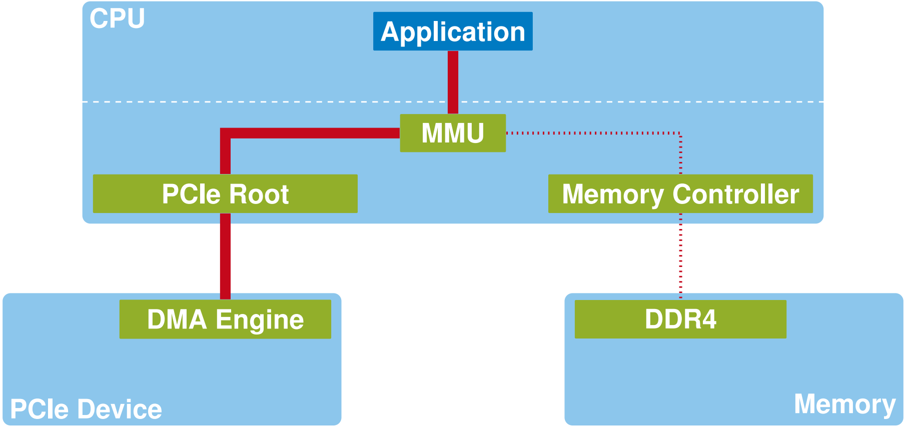
Memory access on modern systems



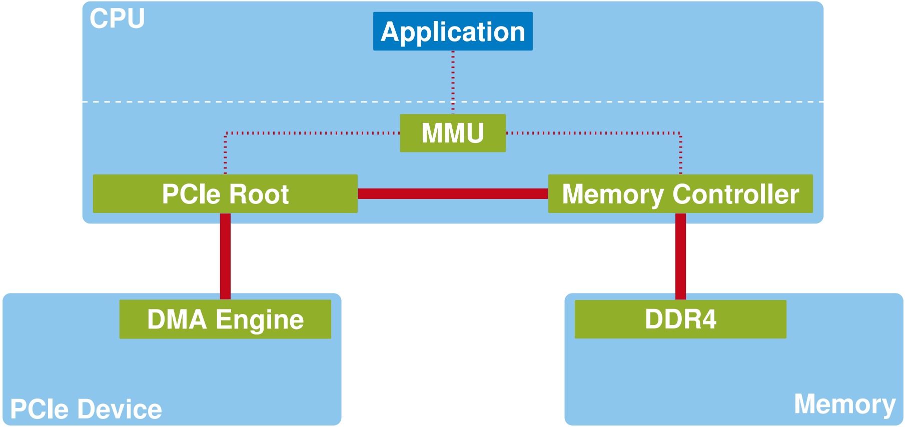
Memory access on modern systems



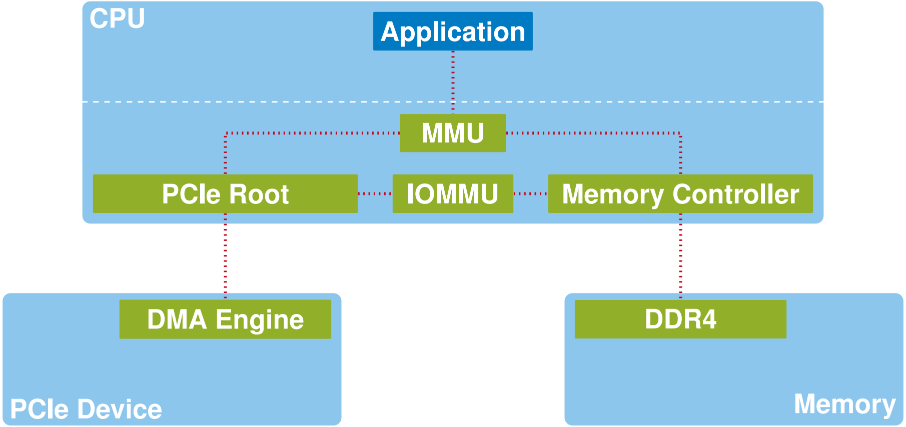
Memory access on modern systems



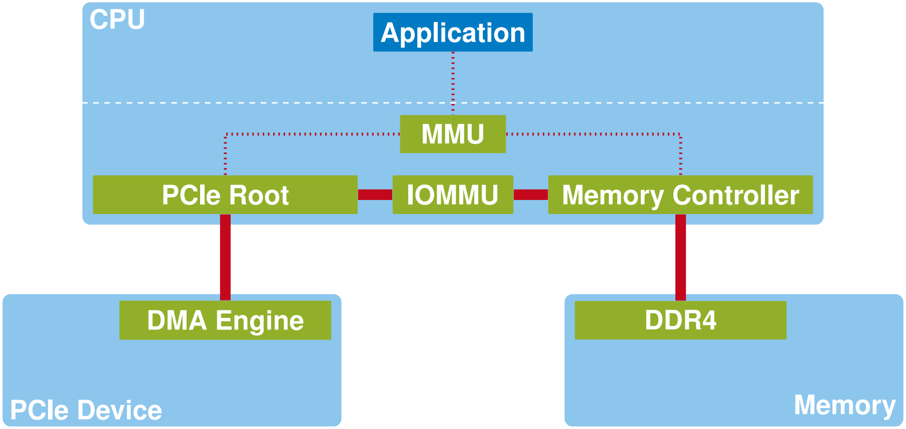
Memory access on modern systems



Memory access on modern systems



Memory access on modern systems


CPU
Application
MMU
PCIe Root
IOMMU
Memory Controller
DMA Engine
PCIe Device
DDR4
Memory