Software Offload for the Masses!

Gianni Antichi

https://gianniantichi.github.io

POLITECNICO MILANO 1863

Queen Mary University of London

BPF offload can accelerate applications

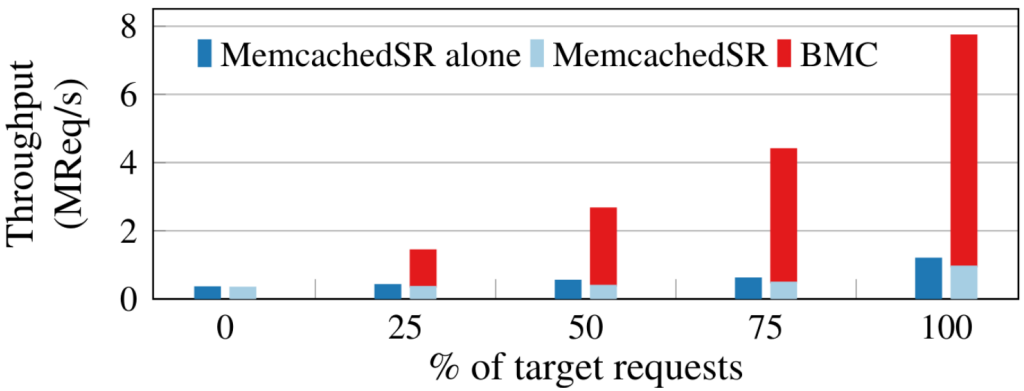
BPF offload can accelerate applications

User Space logic

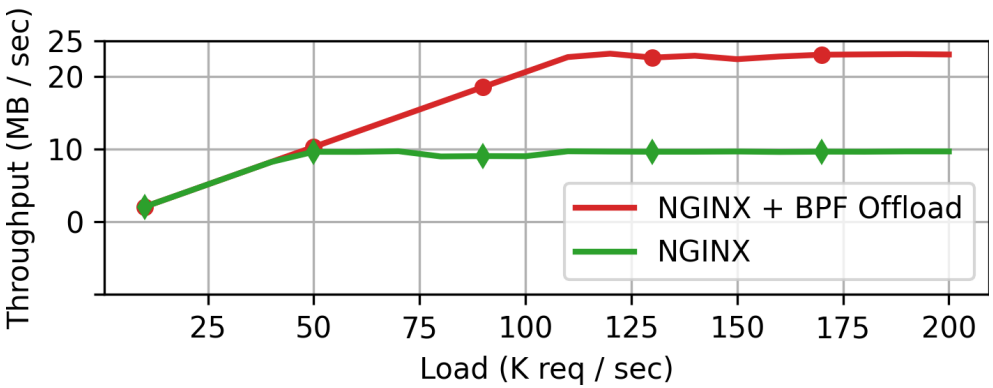
Kernel Space Logic eBPF

- Pushing arbitrary code into kernel without recompile
- Safe (no kernel crash) thanks to the Verifier
- Allow to reduce cycles spent in userspace-kernel transitions during I/O operations

BPF offload can accelerate applications

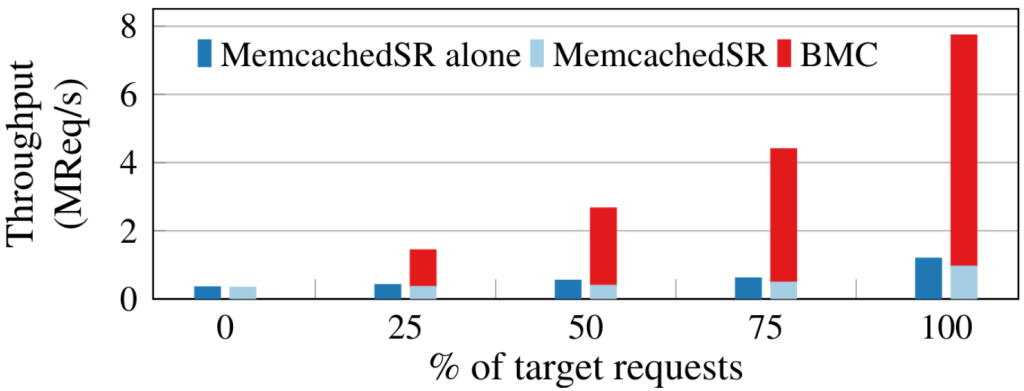


BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing (USENIX NSDI 2021)

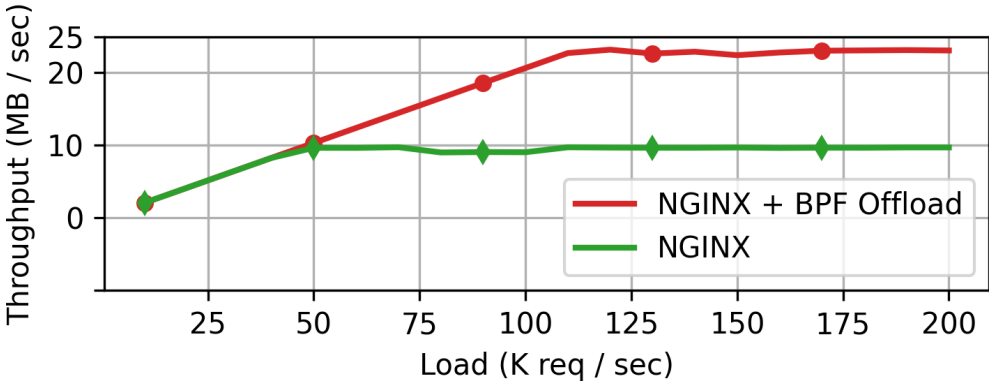


My student ☺

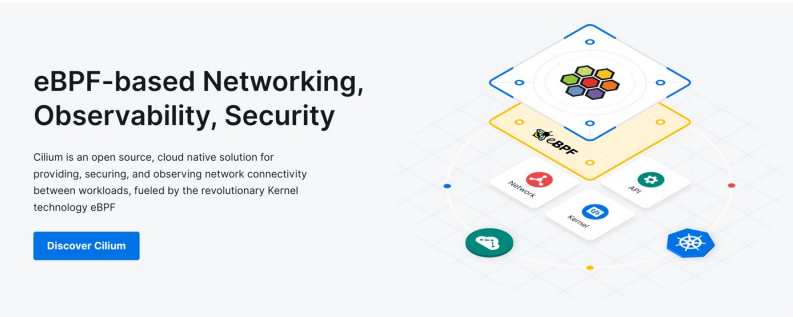
BPF offload can accelerate applications <u>and can be very effective!</u>



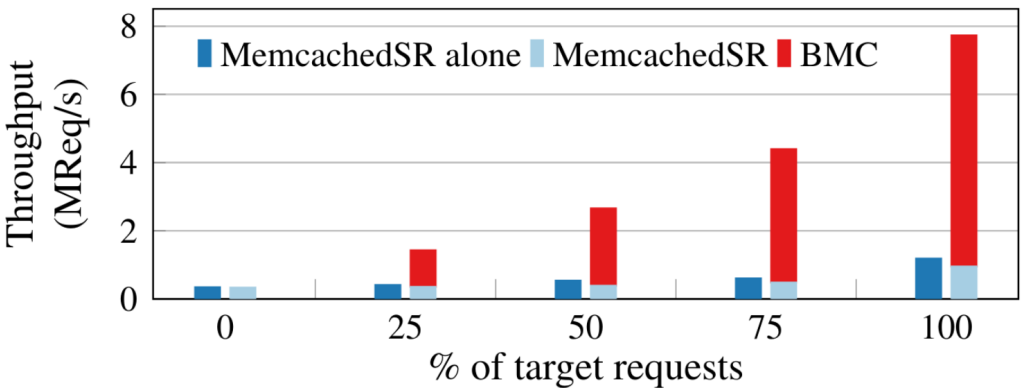
BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing (USENIX NSDI 2021)



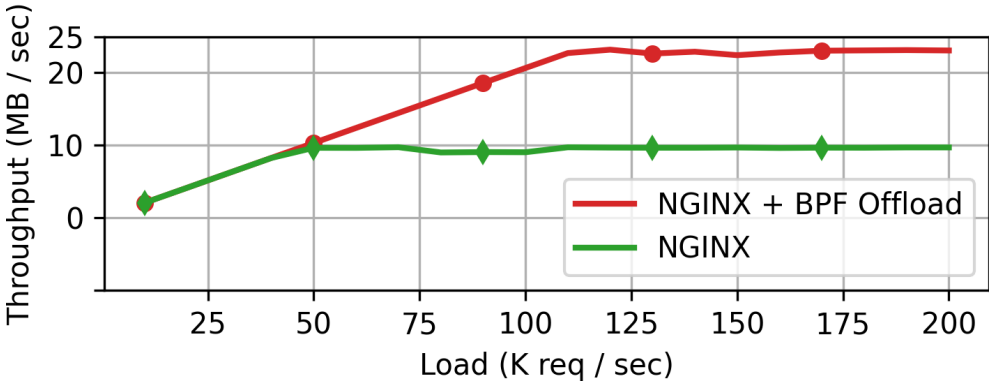
My student ☺



BPF offload can accelerate applications <u>and can be very effective!</u>

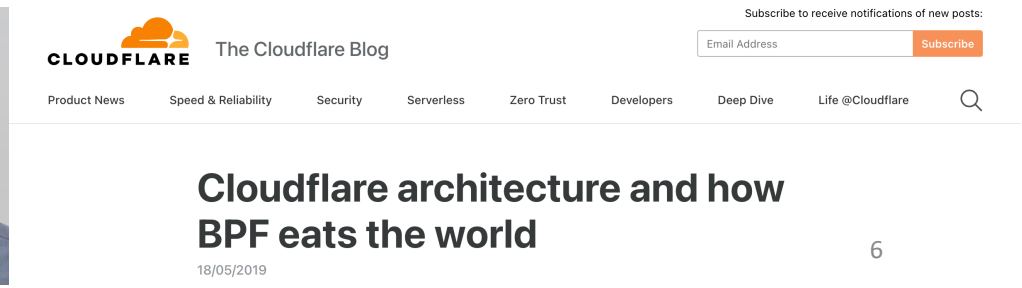


BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing (USENIX NSDI 2021)



My student ☺

Certified!



Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

<u>Current approach:</u>
manually split program execution between “offloadable” code and not

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

<u>Current approach:</u>
manually split program execution between “offloadable” code and not

1. Must understand both application logic and kernel capabilities ☹

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

<u>Current approach:</u>
manually split program execution between “offloadable” code and not

1. Must understand both application logic and kernel capabilities ☹
2. Reason about likely improvements from offloading ☹ ☹

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

<u>Current approach:</u>
manually split program execution between “offloadable” code and not

1. Must understand both application logic and kernel capabilities ☹
2. Reason about likely improvements from offloading ☹ ☹
3. Implement separate user and kernel components ☹ ☹ ☹

Unfortunately..

Problem 1

We cannot offload everything (we need to take into account the restrictions imposed by the kernel verifier)

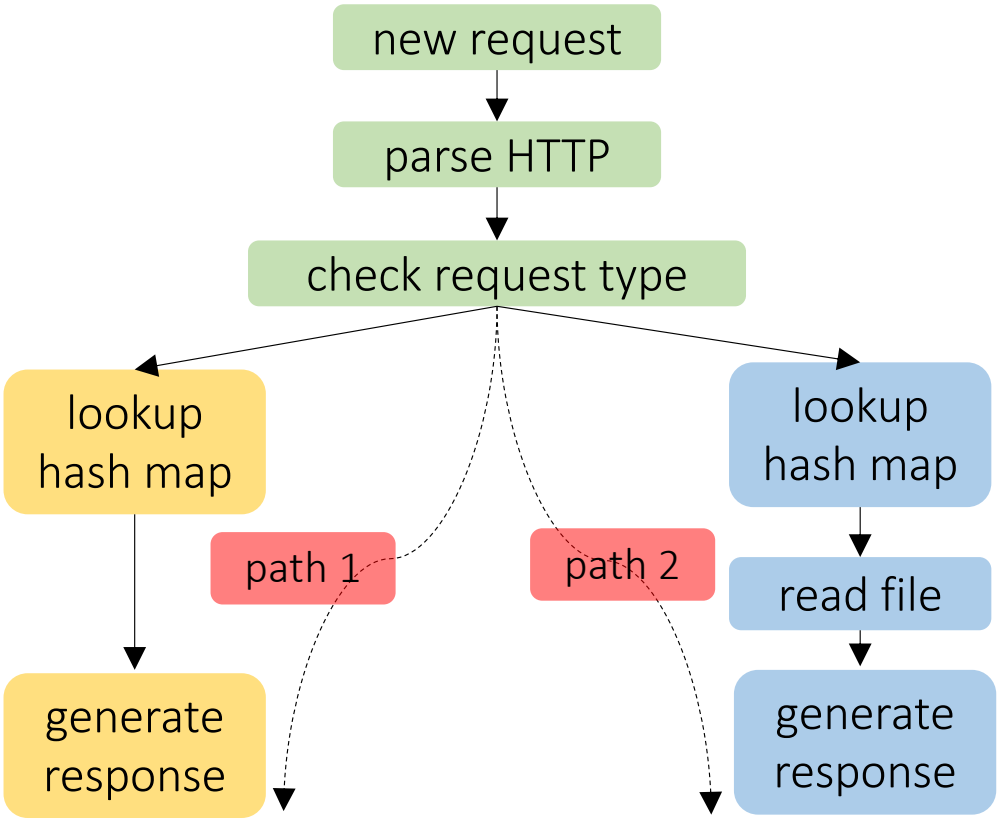
- No task switching (no blocking I/O)
- No complex logic (no floats, no SIMD)
- Limit execution time

<u>Current approach:</u>
manually split program execution between “offloadable” code and not

1. Must understand both application logic and kernel capabilities ☹
2. Reason about likely improvements from offloading ☹ ☹
3. Implement separate user and kernel components ☹ ☹ ☹
4. Can miss opportunities: can only test so many options ☹ ☹ ☹ ☹

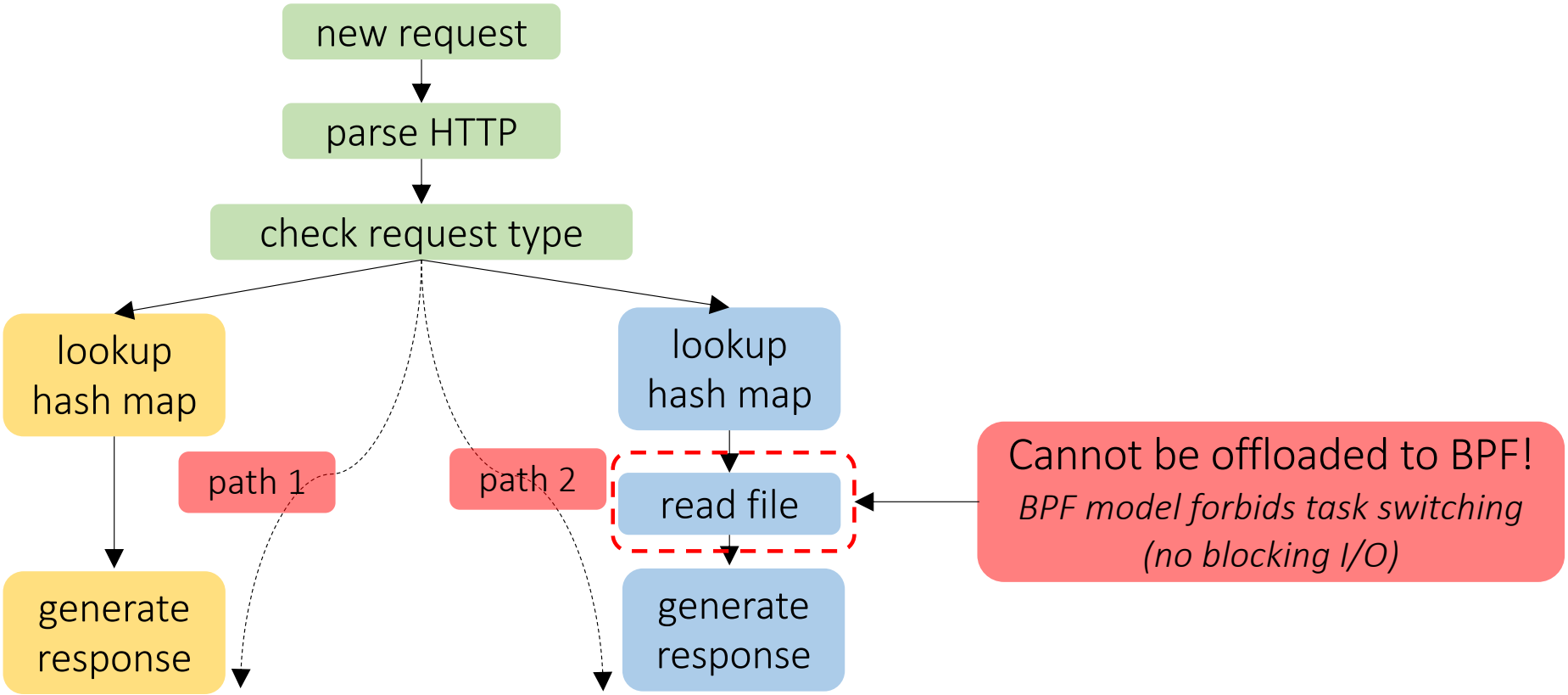
Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results



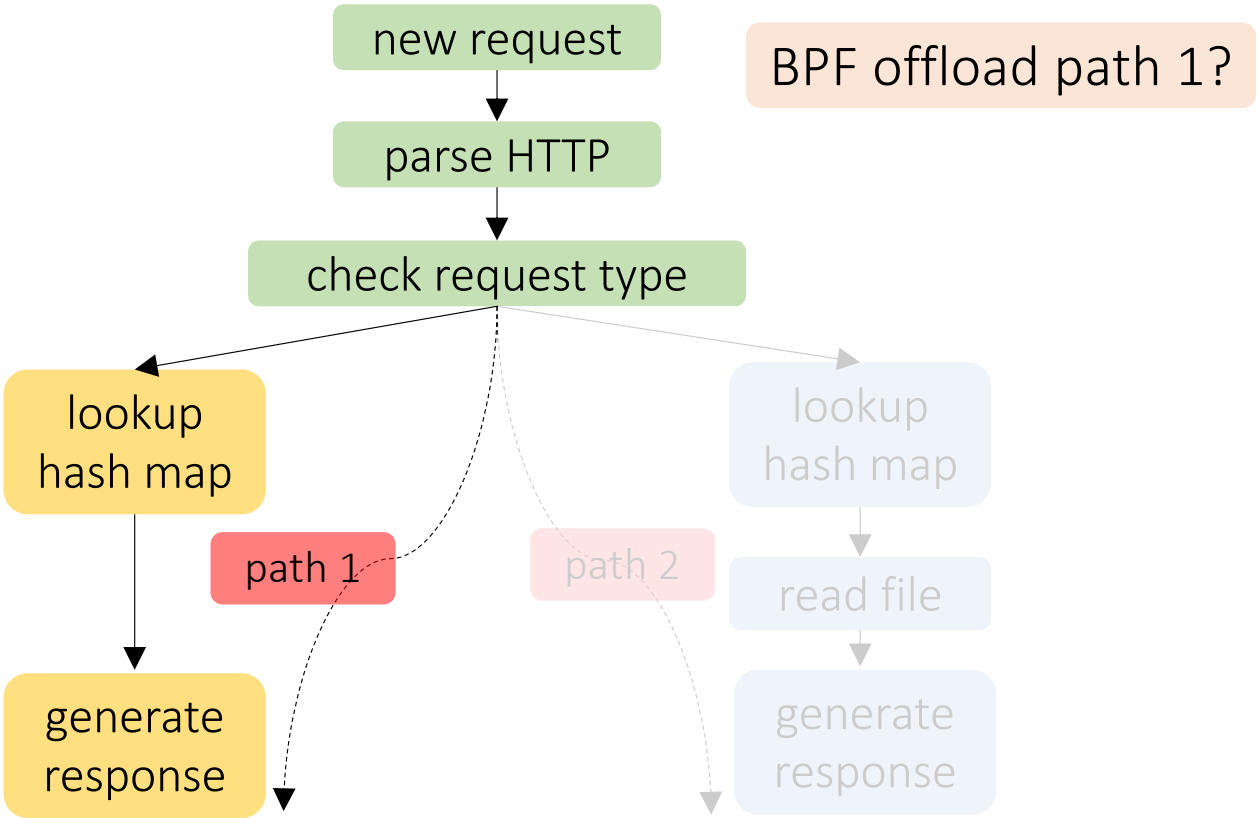
Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results



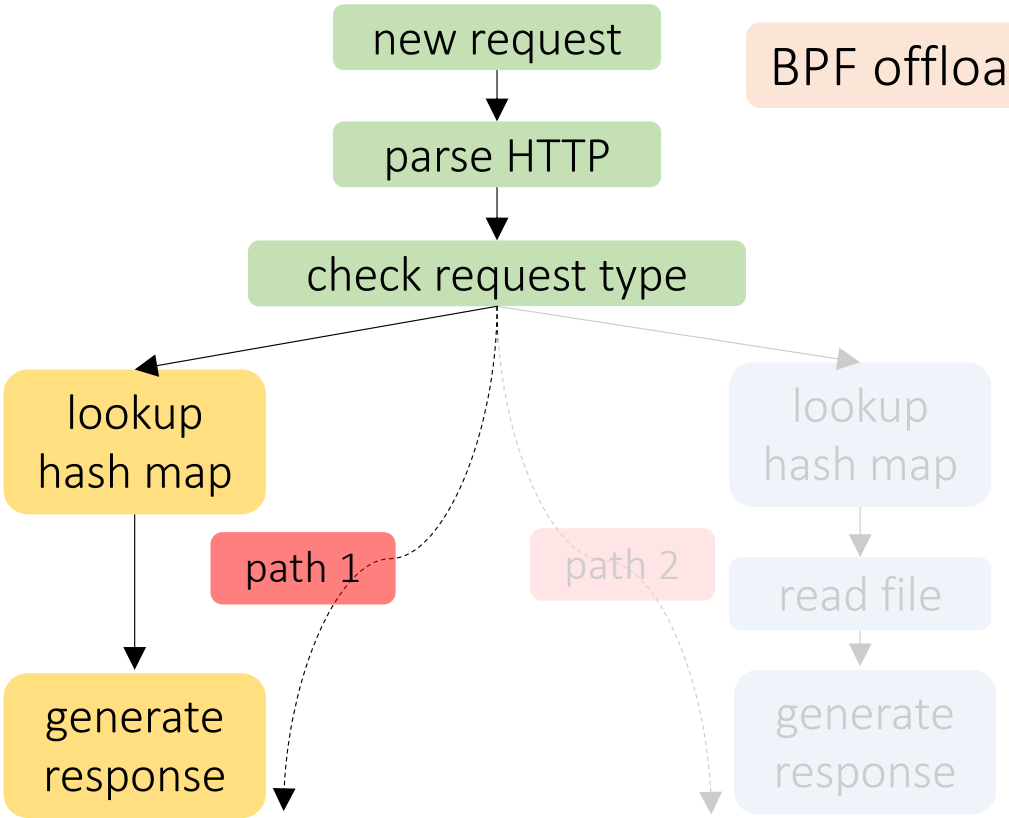
Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results

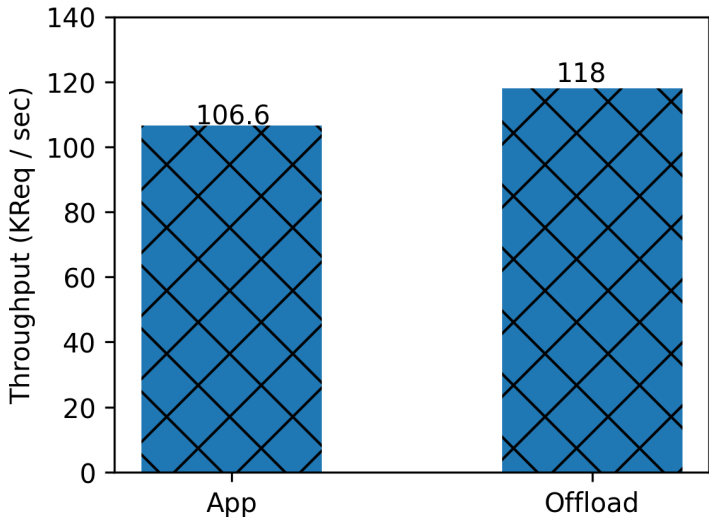


Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results



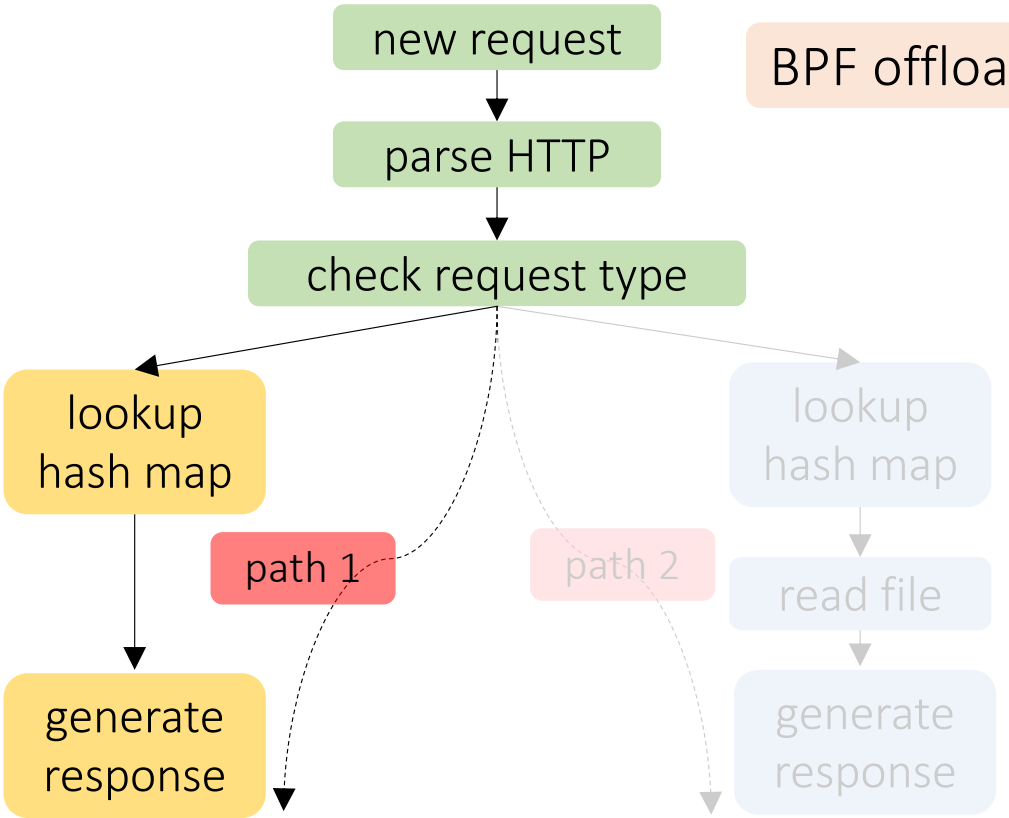
BPF offload path 1?



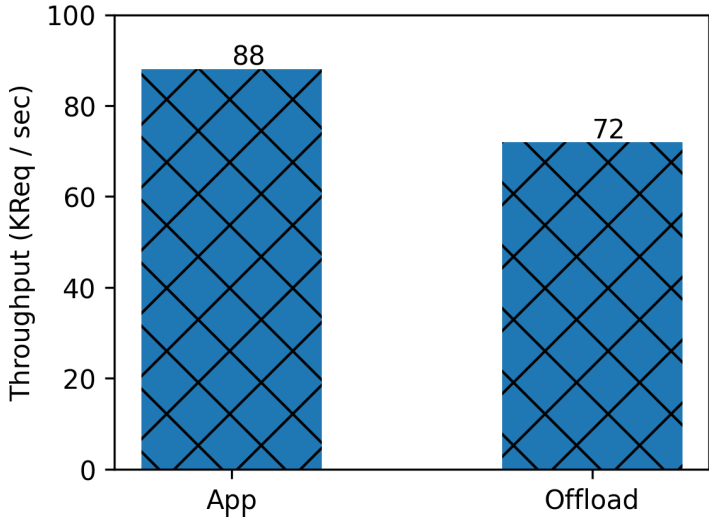
100% hit on path 1

Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results



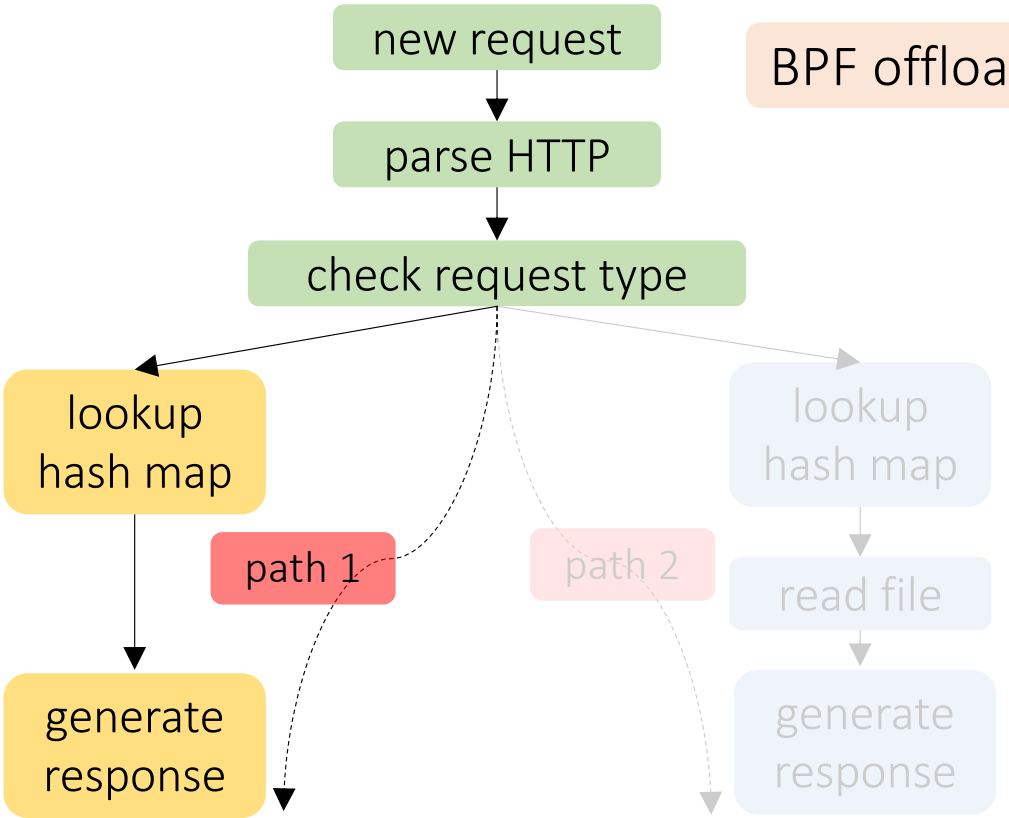
BPF offload path 1?



50% hit on path 1

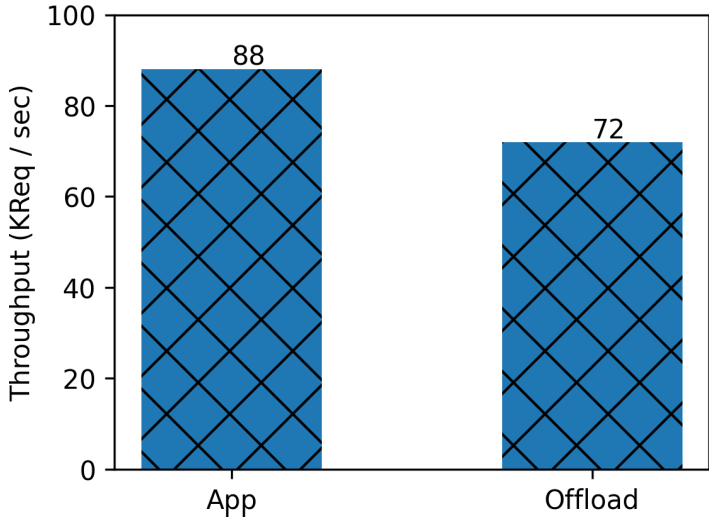
Why this is so hard?

Problem 2 Blindly offload part of programs might lead to unwanted results



BPF offload path 1?

YES! ..but we need to be smart!

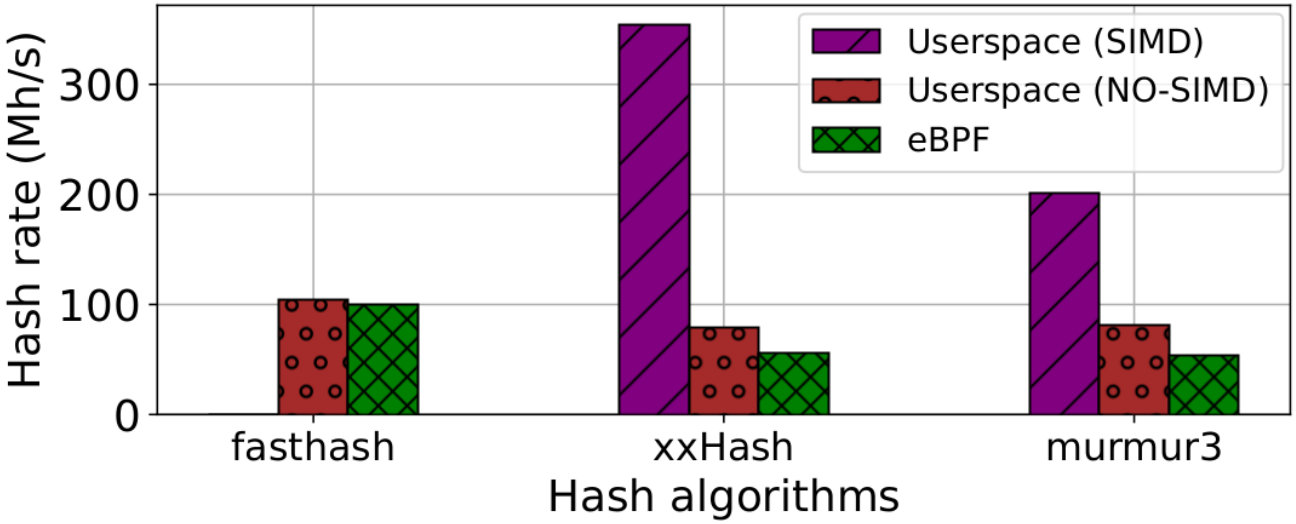


50% hit on path 1

..it gets just worse..

kernel has limitations

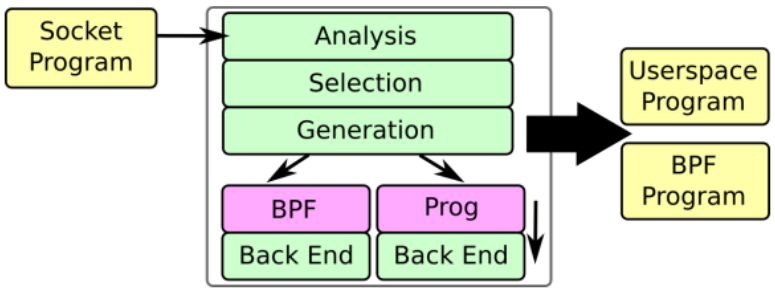
example: no Single Instruction/Multiple Data (SIMD) operations



Fast In-Kernel Sketching with eBPF (ACM Computer Communication Review 2023)

We shall consider this when deciding *what to offload*

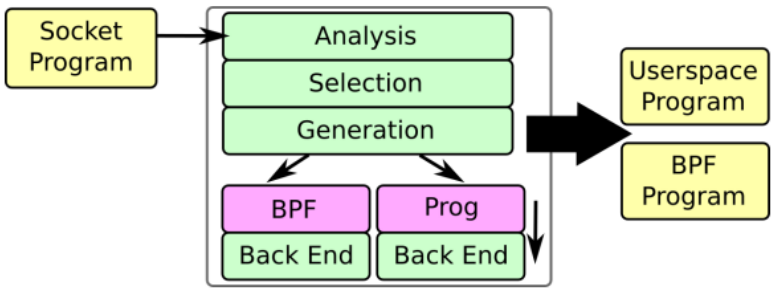
Our proposal: Automate Offloading



Similar to approaches adopted by other accelerators (e.g., ML accelerators, video encoders)

Automatic Kernel Offload Using BPF. F. Shahinfar, S. Miano, G. Siracusano, R. Bifulco, A. Panda, G. Antichi. ACM HotOS 2023

Our proposal: Automate Offloading

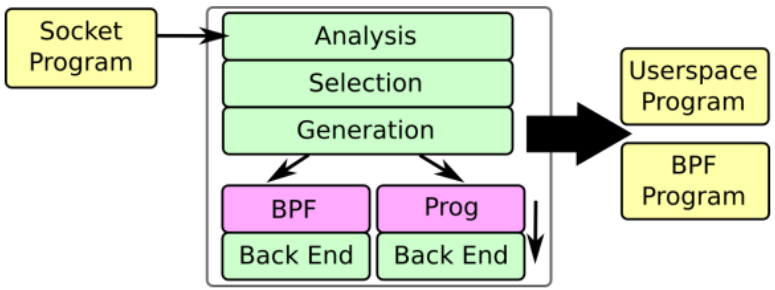


Similar to approaches adopted by other accelerators (e.g., ML accelerators, video encoders)

<u>What it is different:</u>
offload has no new processing capabilities

Automatic Kernel Offload Using BPF. F. Shahinfar, S. Miano, G. Siracusano, R. Bifulco, A. Panda, G. Antichi. ACM HotOS 2023

Our proposal: Automate Offloading



Similar to approaches adopted by other accelerators (e.g., ML accelerators, video encoders)

What it is different:
offload has no new processing capabilities

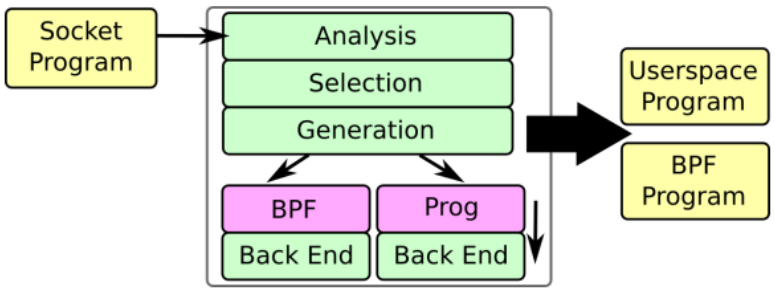
1 Analyze

We need to slice the program rather than identify calls to common functions

program slicing

Automatic Kernel Offload Using BPF. F. Shahinfar, S. Miano, G. Siracusano, R. Bifulco, A. Panda, G. Antichi. ACM HotOS 2023

Our proposal: Automate Offloading



Similar to approaches adopted by other accelerators (e.g., ML accelerators, video encoders)

<u>What it is different:</u>
offload has no new processing capabilities

1 Analyze

2 Partition

- We need to devise coordination mechanisms
  - ✓ kernel offload vs user space program
- We need cost models
  - ✓ Is the offload worth it?

program slicing

+

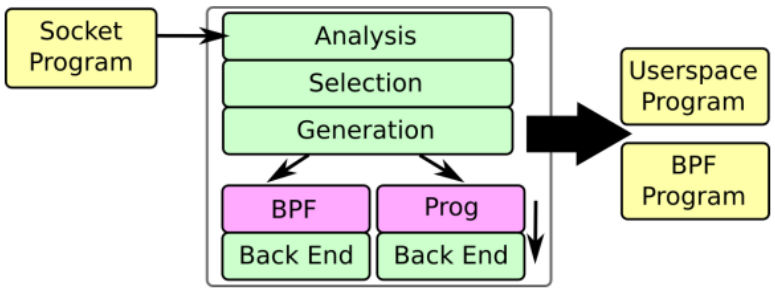
PGO/FDO

+

models

Automatic Kernel Offload Using BPF. F. Shahinfar, S. Miano, G. Siracusano, R. Bifulco, A. Panda, G. Antichi. ACM HotOS 2023

Our proposal: Automate Offloading



Similar to approaches adopted by other accelerators (e.g., ML accelerators, video encoders)

What it is different:
offload has no new processing capabilities

1. Analyze
2. Partition
3. Compile

Kernel changes often: a core difference between offloading to software vs hardware

program slicing
+
PGO/FDO
+
models

Automatic Kernel Offload Using BPF. F. Shahinfar, S. Miano, G. Siracusano, R. Bifulco, A. Panda, G. Antichi. ACM HotOS 2023

Final considerations

- Automate BPF offload is possible but hard ☺
- Working on a first prototype with encouraging results
- Shall we add support for common operations on Linux?
  - Example: kTLS
  - What *common* means though?

User Space logic

Kernel Space Logic

A special thanks to…

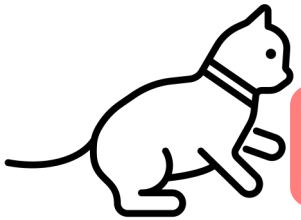
Farbod Shahinfar

Sebastiano Miano

Giuseppe Siracusano

Roberto Bifulco

Aurojit Panda

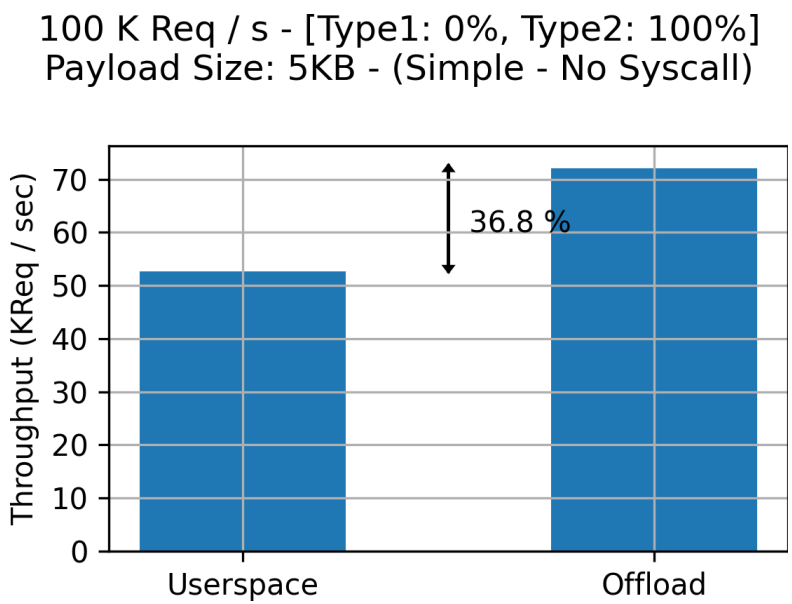


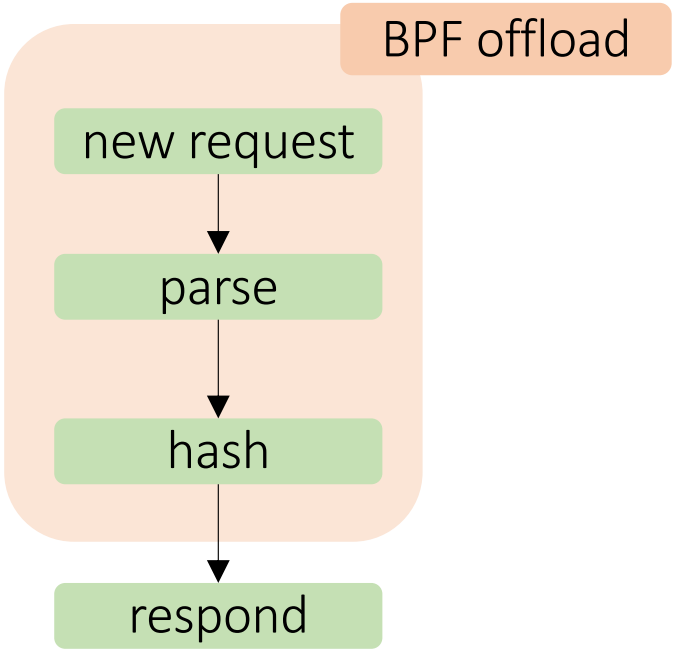
Questions?

Backup

A last remark: it is not only about logic offload

Data summarization seems promising





A last remark: it is not only about logic offload

Data summarization seems promising

...but syscalls can kill a bit the party ☺