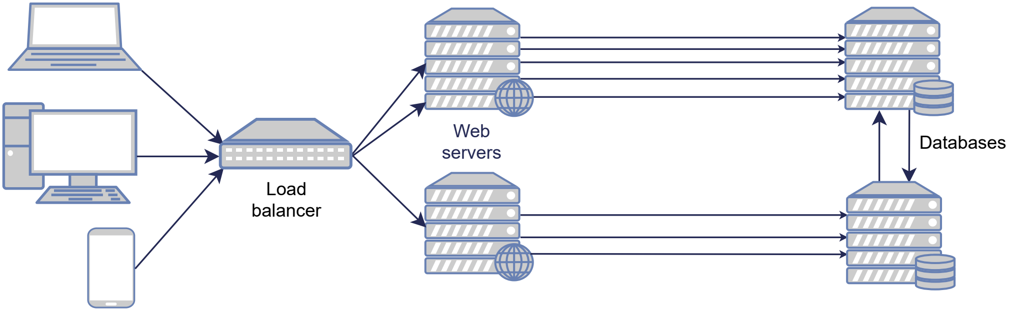
# BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing

Yoann Ghigoff[1,2,3] Julien Sopena[2] Kahina Lazri[1] Antoine Blin[4] Gilles Muller[3]

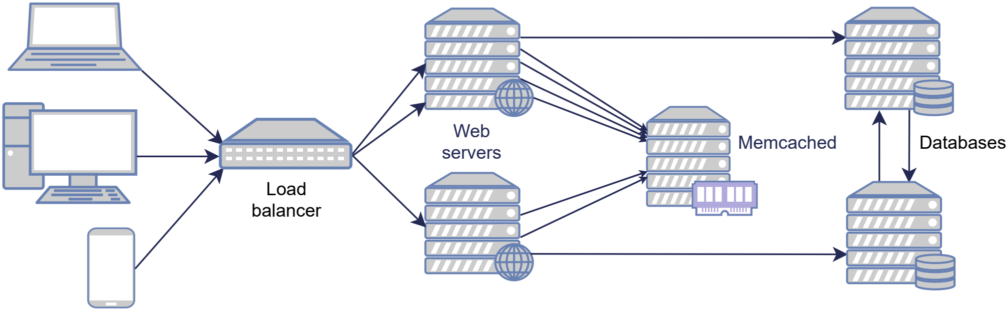
[1]Orange Labs [2]LIP6 – Sorbonne University [3]Inria [4]Gandi

*Networked Systems Design and Implementation, 12–14 April, 2021*

# Memcached



# Memcached

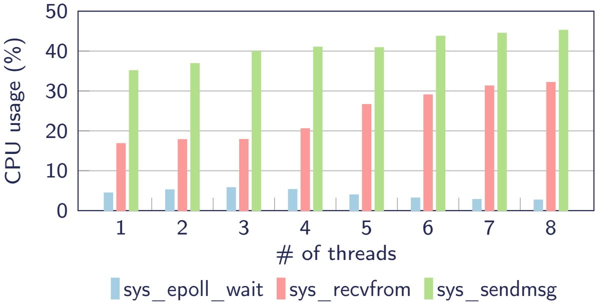



- In-memory key-value store, simple GET/SET interface
- Used by web services to off load work from databases
- Given its crucial role, Memcached must be able to sustain high network load

# Observations

Memcached suffers from known kernel overheads:

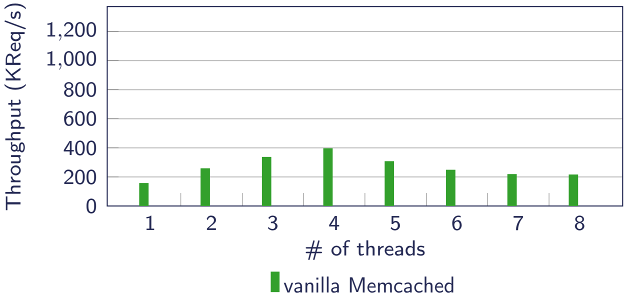
- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls



# Observations

Memcached suffers from known kernel overheads:

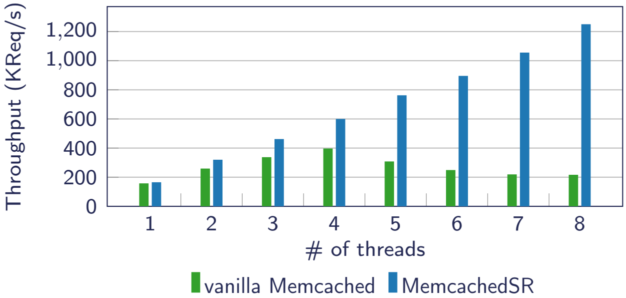
- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls



# Observations

Memcached suffers from known kernel overheads:

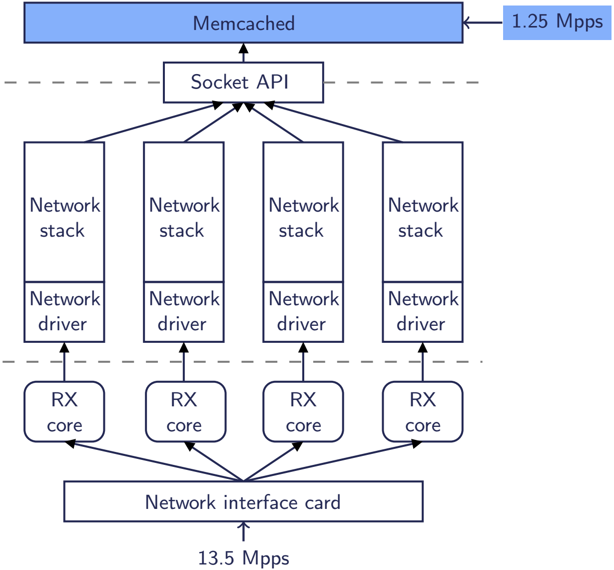
- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls



# Observations

Memcached suffers from known kernel overheads:

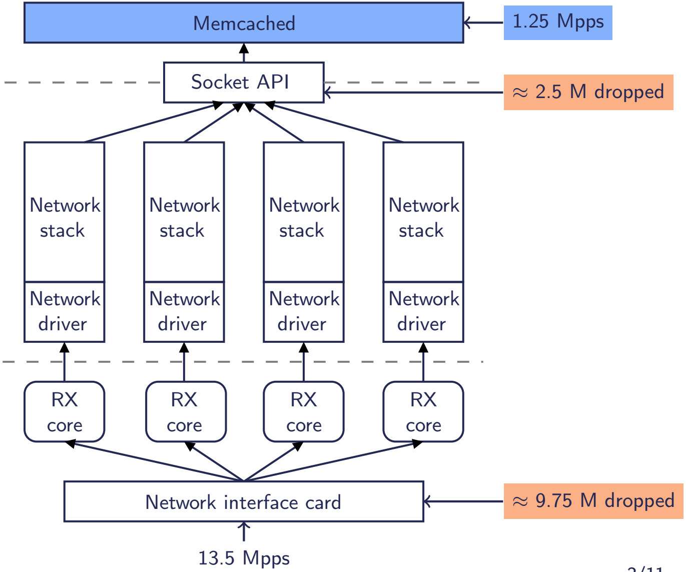
- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls
- Per-packet TCP/IP processing



# Observations

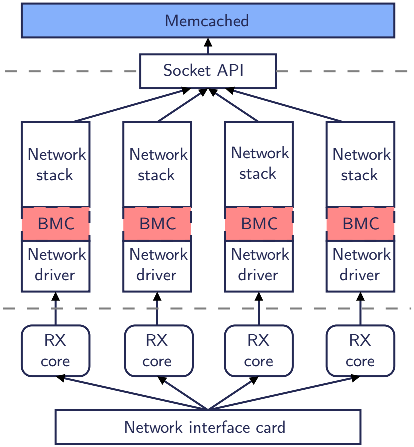
Memcached suffers from known kernel overheads:

- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls
- Per-packet TCP/IP processing



# BMC: Key idea

Enable a Memcached server to respond to *get* requests without executing the whole network stack

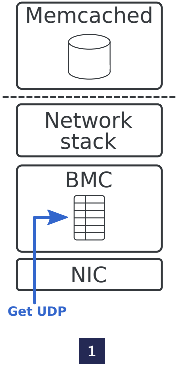


# BMC: in-kernel proxy

Enable a Memcached server to respond to *get* requests without executing the whole network stack

Filters memcached packets in order to:

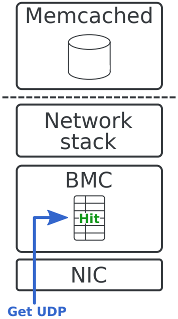
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application

# BMC: in-kernel proxy

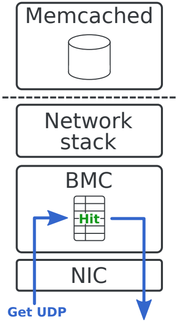



1

Filters memcached packets in order to:

1. Serve get requests on behalf of the application

# BMC: in-kernel proxy

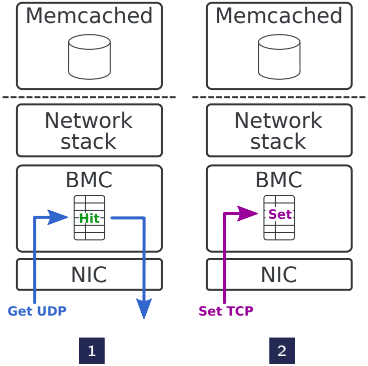



1

Filters memcached packets in order to:

1. Serve get requests on behalf of the application

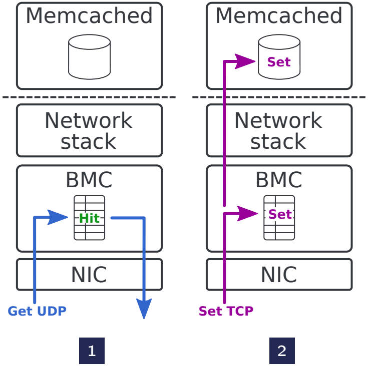
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

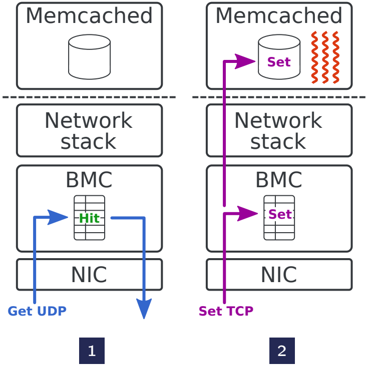
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

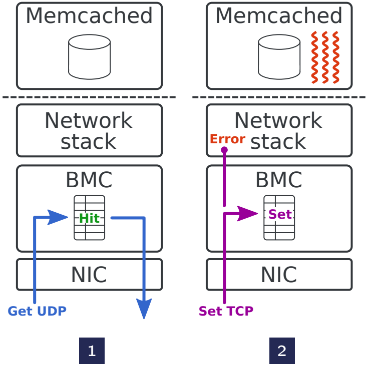
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

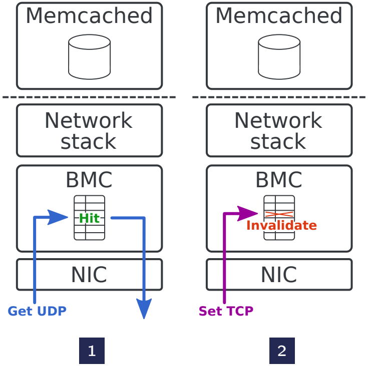
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

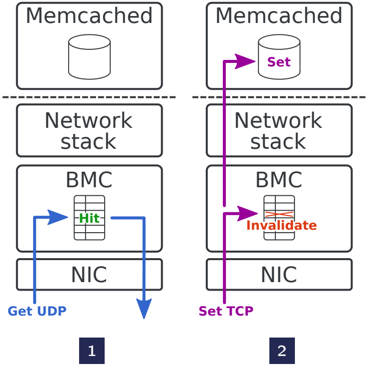
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

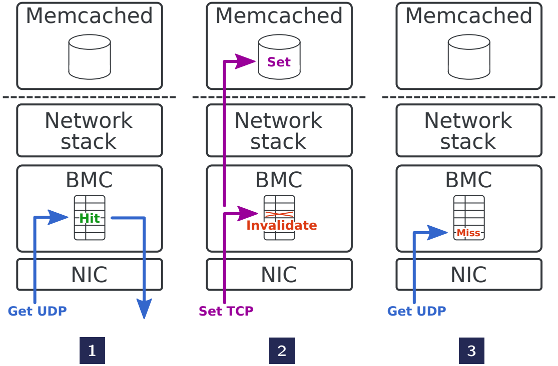
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible

# BMC: in-kernel proxy



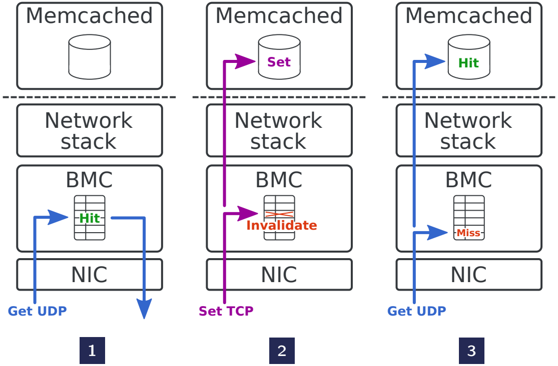

Filters memcached packets in order to:

1 Serve get requests on behalf of the application

2 Ensure cache coherence as simply as possible

3 Perform cache updates transparently to the application

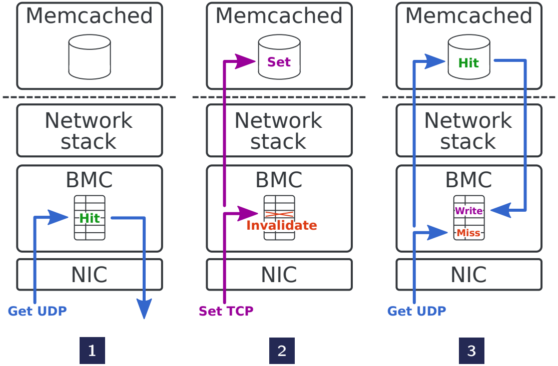
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application

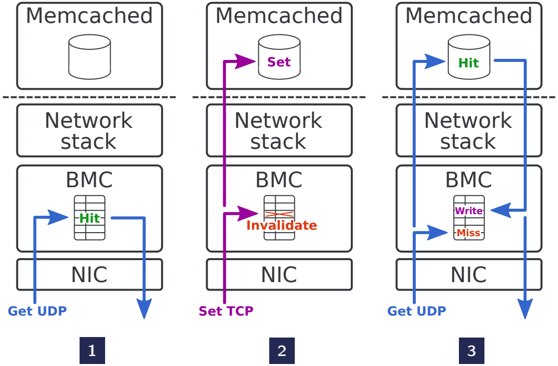
# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application

# BMC: in-kernel proxy




Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application

# Implementation

## eBPF

Linux's extended version of BPF. The eBPF infrastructure offers the ability to run user-supplied programs inside the kernel.

## Benefits of eBPF

- Safety through static analysis
- JIT compilation
- Network driver attach point (XDP)

# Implementation

## eBPF

Linux's extended version of BPF. The eBPF infrastructure offers the ability to run user-supplied programs inside the kernel.

## Limitations of eBPF

Static analysis doesn't scale to complex application logic

- Only a limited number of BPF instructions can be analyzed
- Loops must have static bounds
- No dynamic memory allocation

# Dealing with eBPF's limitations

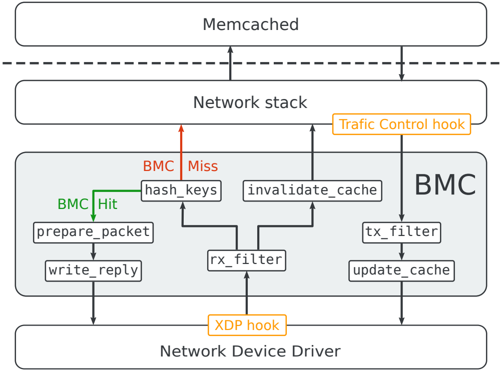
- Bounding data (memcached keys/data/packets)

# Dealing with eBPF's limitations

- Bounding data (memcached keys/data/packets)
- Using a rolling hash function (*FNV-1a*)

# Dealing with eBPF's limitations

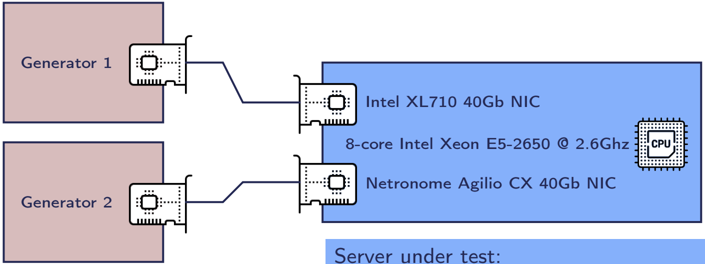
- Bounding data (memcached keys/data/packets)
- Using a rolling hash function (*FNV-1a*)
- Partitioning complex functions



# Experimental setup

Two clients generate memcached worload:

- 100 million distinct memcached keys
- Zipf key distribution
- 16-byte keys and 32-byte values
- 30:1 GET/SET ratio

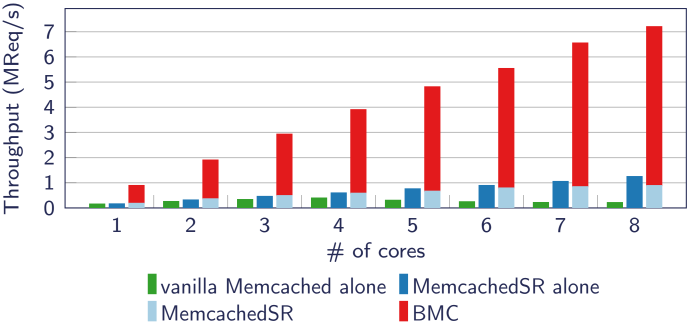



Server under test:

- Linux 5.3.0
- Memcached 1.5.19 with 10 GB of memory
- BMC with 2.5 GB of memory

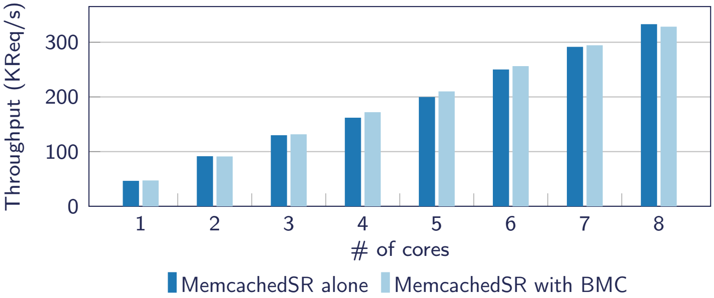
# Throughput

- Up to 18x compared to vanilla Memcached
- Up to 6x compared to MemcachedSR



# Throughput

- Up to 18x compared to vanilla Memcached
- Up to 6x compared to MemcachedSR
- No observable deterioration with a worst-case workload



# Receive-Process-Reply latency

- Median of memcached hits and misses with BMC is respectively 21.8 and 21.6 µs
- 2.11 µs for a BMC cache hit
- Memcached operations are about 1 µs faster when not running BMC

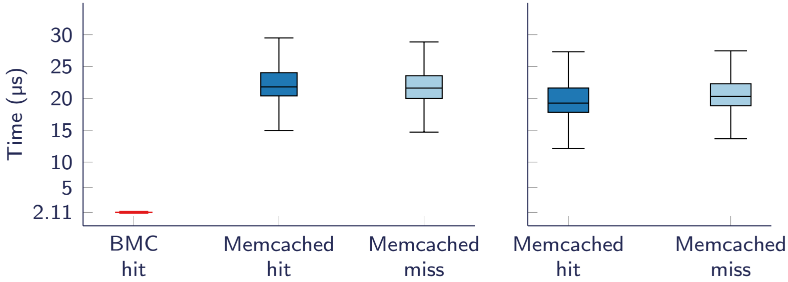
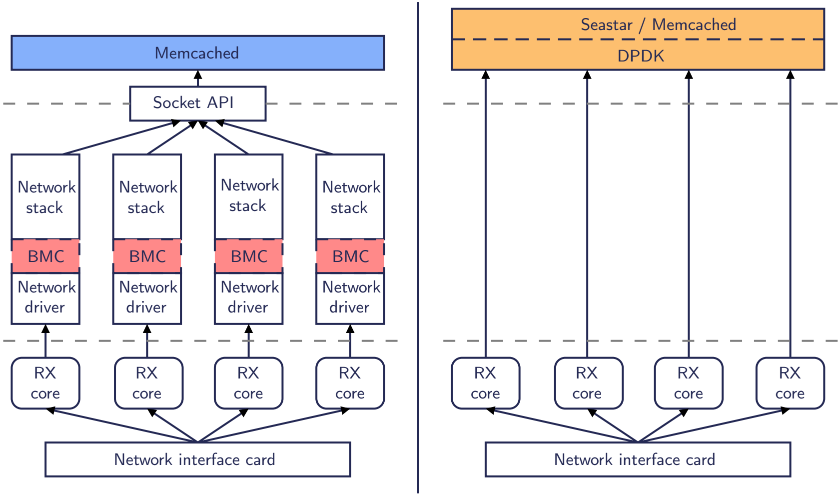



Fig 1: with BMC

Fig 2: without BMC

# Comparison to kernel-bypass: Seastar



# Comparison to kernel-bypass: throughput

- Up to 5x higher throughput on favorable workload
- Better performance scaling on mixed workload

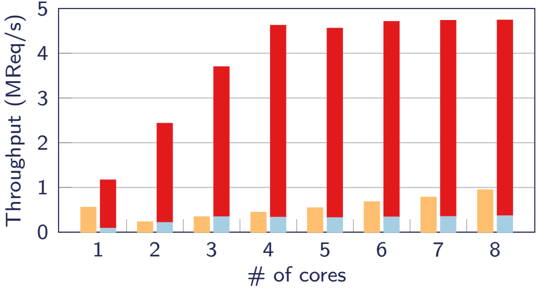



Fig 1: UDP only

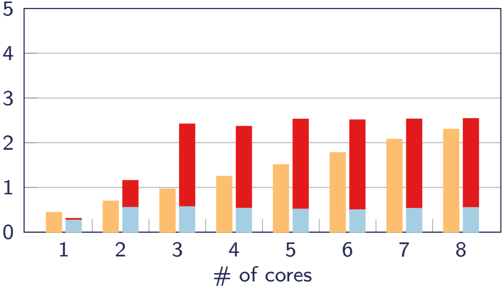
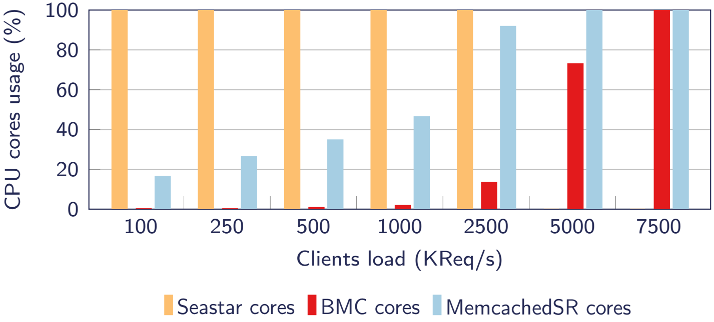



Fig 2: UDP and TCP

Seastar MemcachedSR BMC

# Comparison to kernel-bypass: CPU usage

- Up to 5x higher throughput on favorable workload
- Better performance scaling on mixed workload
- 3x times less CPU resources to achieve similar throughput



# Conclusion

BMC

- uses in-kernel caching to serve Memcached requests after they have been received by the network driver
- works with unmodified software on commodity hardware
- offers significant throughput improvement
- introduces negligible overhead

**On-going work:** Optimized eviction algorithm

# Thank you

For more questions:

yoann.ghigoff@inria.fr