Performance evaluation of Linux CAN-related system calls

Michal Sojka, Pavel Píša, Zdeněk Hanzálek

Czech Technical University in Prague,
Faculty of Electrical Engineering
Email: {sojkam1,pisa,hanzalek}@fel.cvut.cz

Workshop on Factory Communication Systems (WFCS)
May 7, 2014 – Toulouse, France

Motivation

Security of on-board networks

Motivation

Security of on-board networks

Intrusion detection system

Motivation

Security of on-board networks

Intrusion detection system

↳ Needs to be "tuned" to avoid false positives

Motivation

Security of on-board networks

Intrusion detection system

↳ Needs to be "tuned" to avoid false positives

↳ Data from outdoor experiments

Motivation

Security of on-board networks

Intrusion detection system

- ↳ Needs to be "tuned" to avoid false positives
  - ↳ Data from outdoor experiments
    - ↳ **Logging**

Motivation

Security of on-board networks

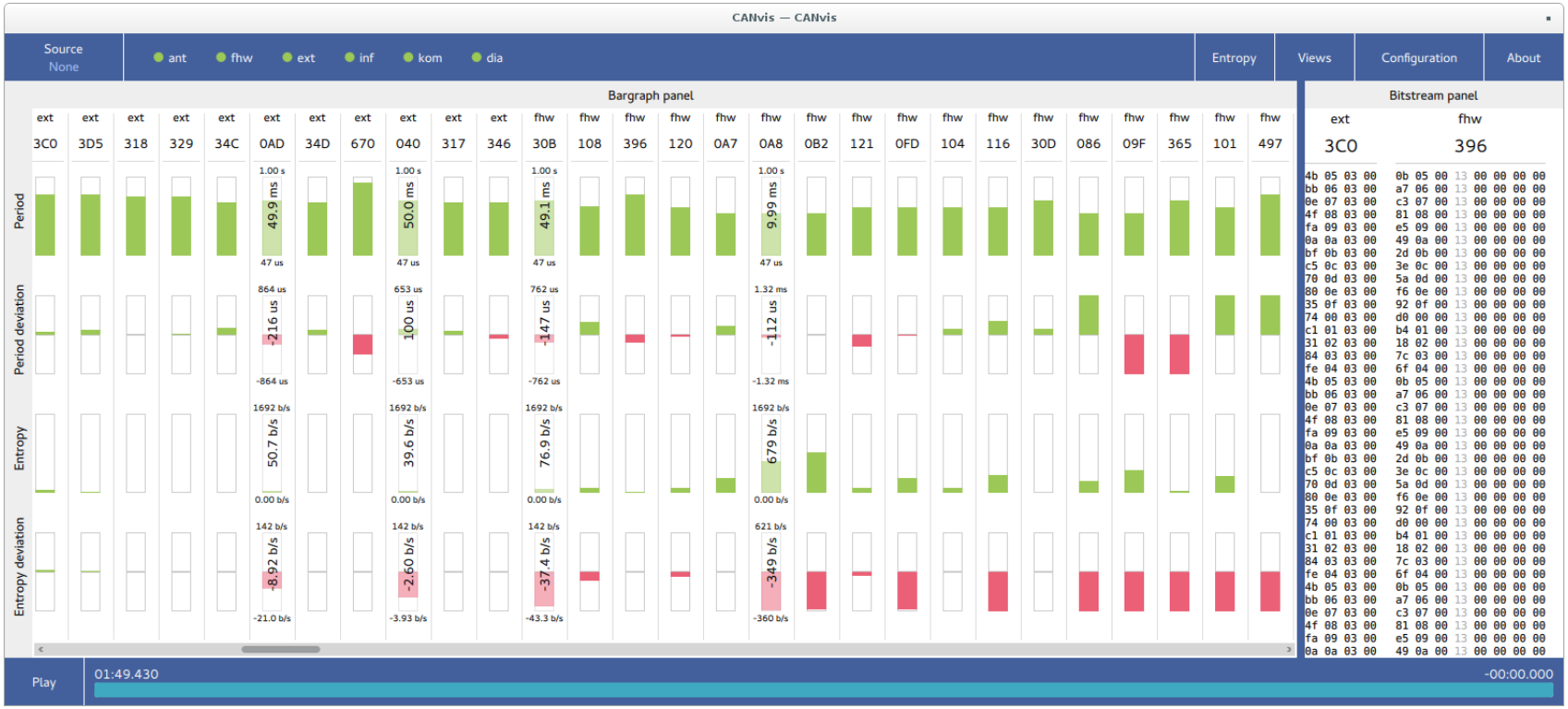
Intrusion detection system

↳ Needs to be "tuned" to avoid false positives

↳ Data from outdoor experiments

↳ **Logging**

Requirements



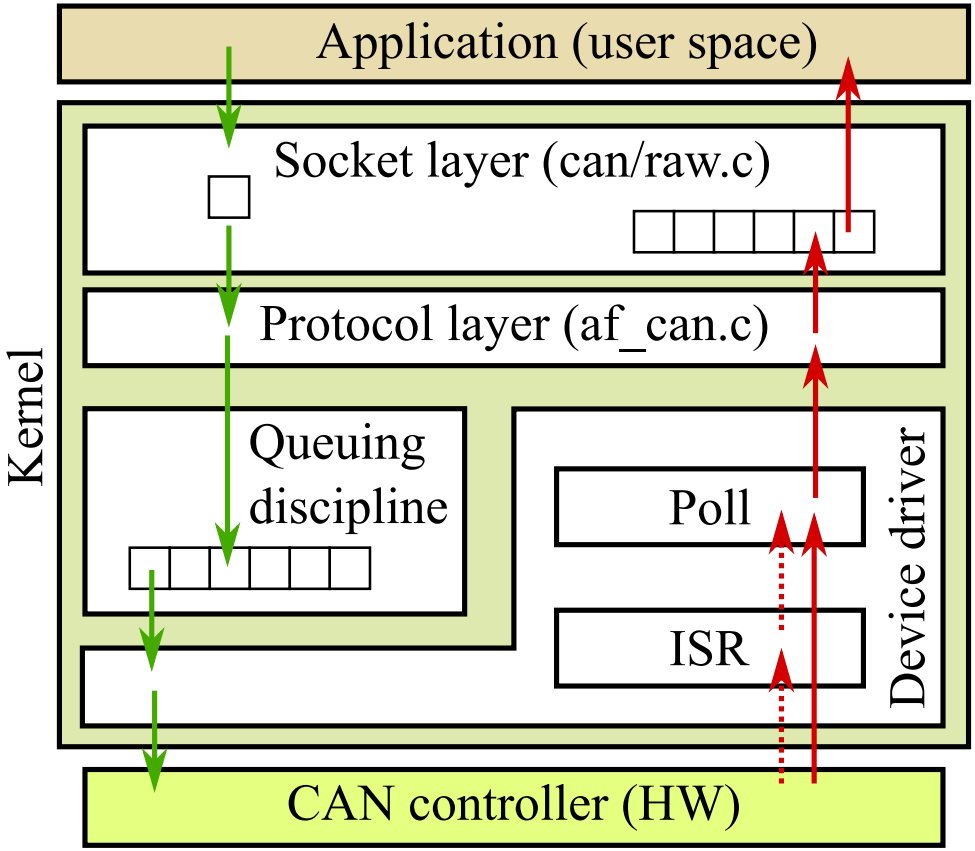
- All CAN traffic from a car has to be logged in real-time.
- Up to 20 CAN buses needs to be logged simultaneously.
- Logs are analyzed off-line.

Outline

Linux networking subsystem

RX/TX path overheads:

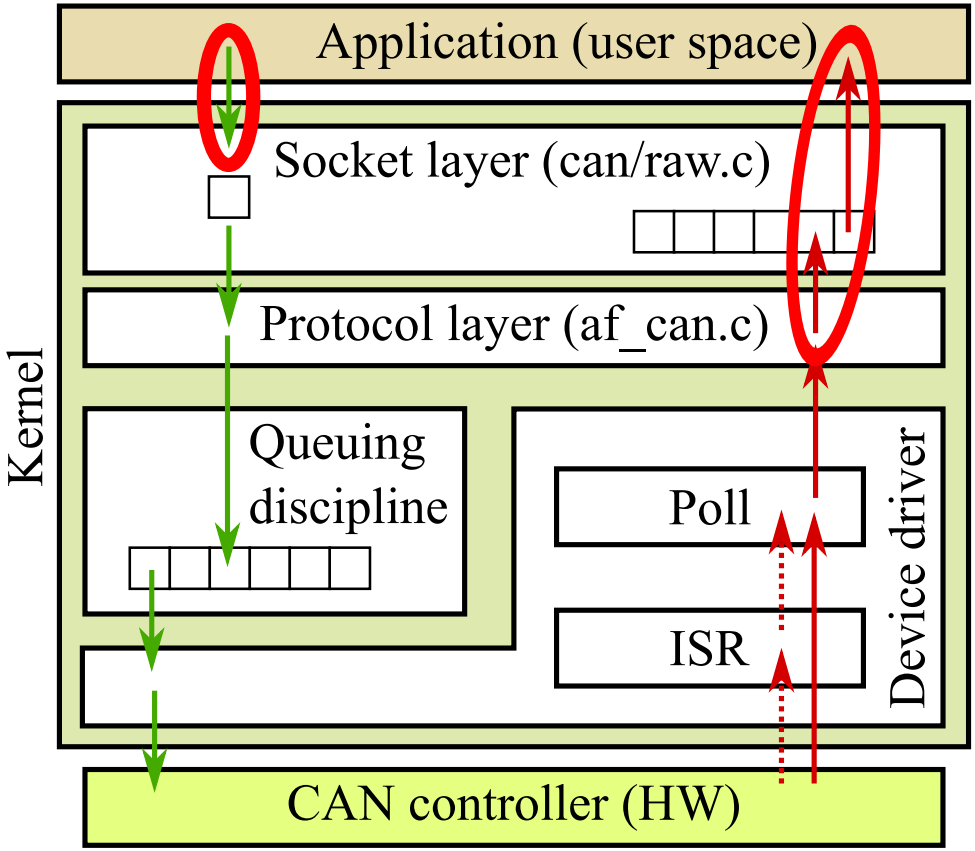
- Interrupt
- Socket, protocol, Qdisc
- System call
- We are interested in the interface between applications and the kernel.



Linux networking subsystem

RX/TX path overheads:

- Interrupt
- Socket, protocol, Qdisc
- System call
- We are interested in the interface between applications and the kernel.



Linux networking API

Sockets:

- Raw CAN sockets – most common
- Memory mapped socket (PF_PACKET)
- Non-blocking I/O (O_NONBLOCK)
- New "Low-latency" sockets (Linux 3.11 – 9/2013)

Operations on sockets:

- read/write (send/recv)
- sendmmsg/recvmmsg

Memory-mapped sockets

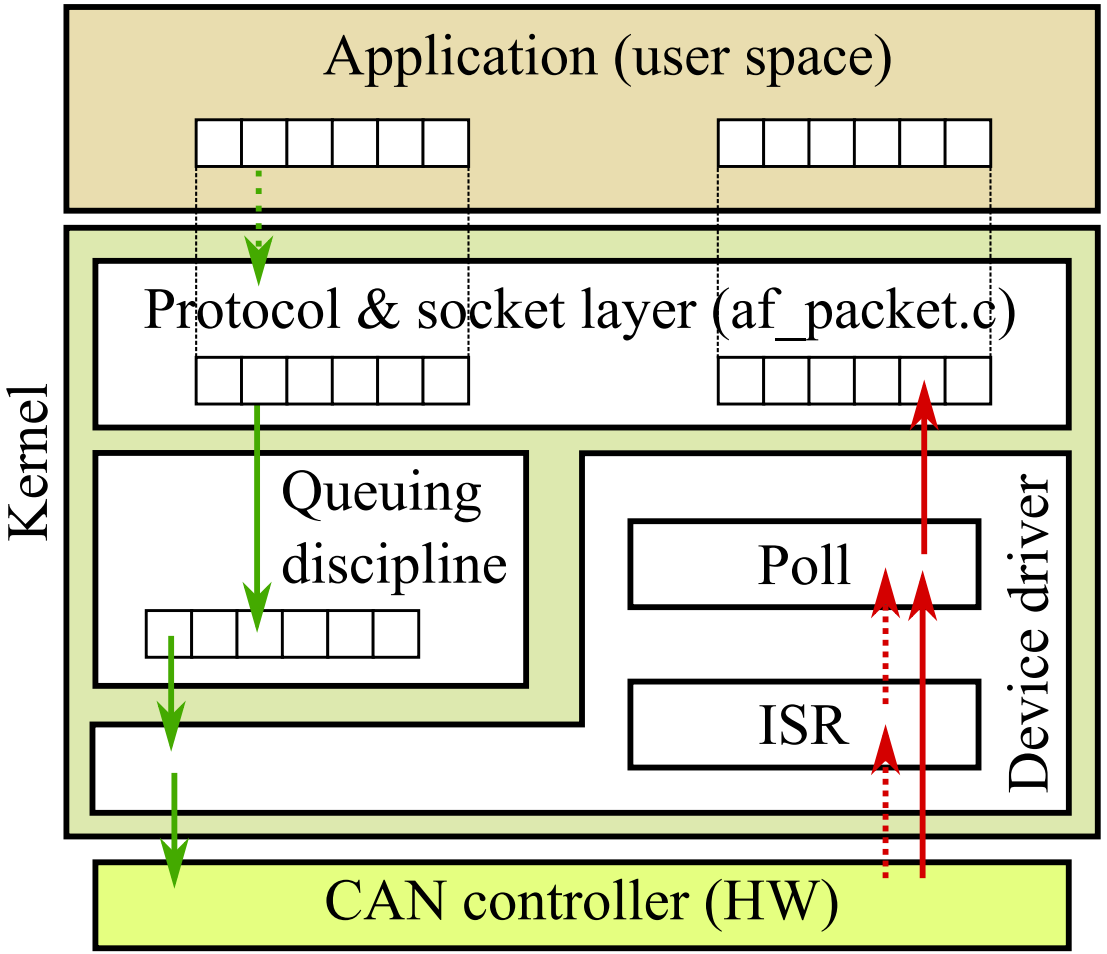
PF_PACKET

Reception

- no system call
- wait: `poll()`

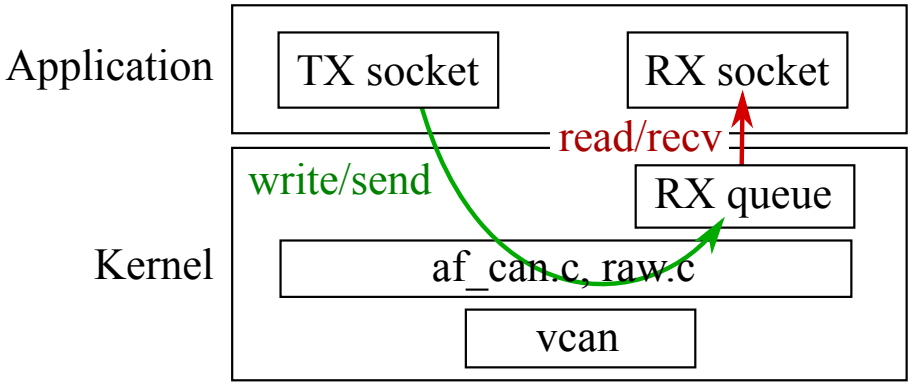
Transmission

- Notify the kernel about ready messages.



Single/multiple messages per system call

- Comparison of `recv()` and `recvmmsg()` system calls
- Use of virtual CAN interface (vcan)

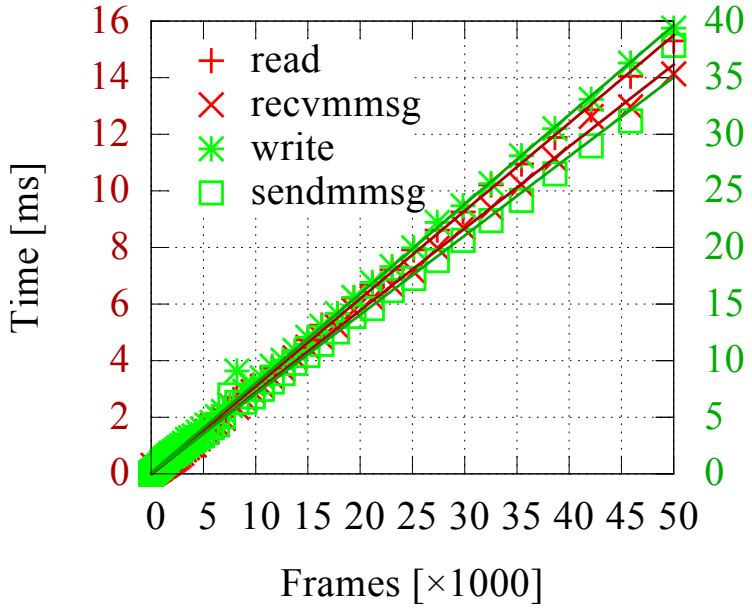


Experiment

1. Send X messages – kernel queues them in an RX queue
2. Receive X messages from the RX queue

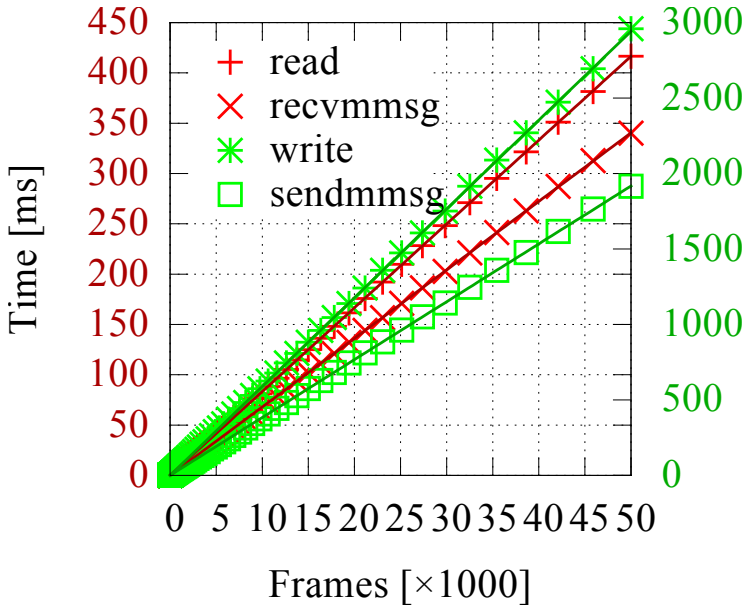
With or without mmsg – results

PC



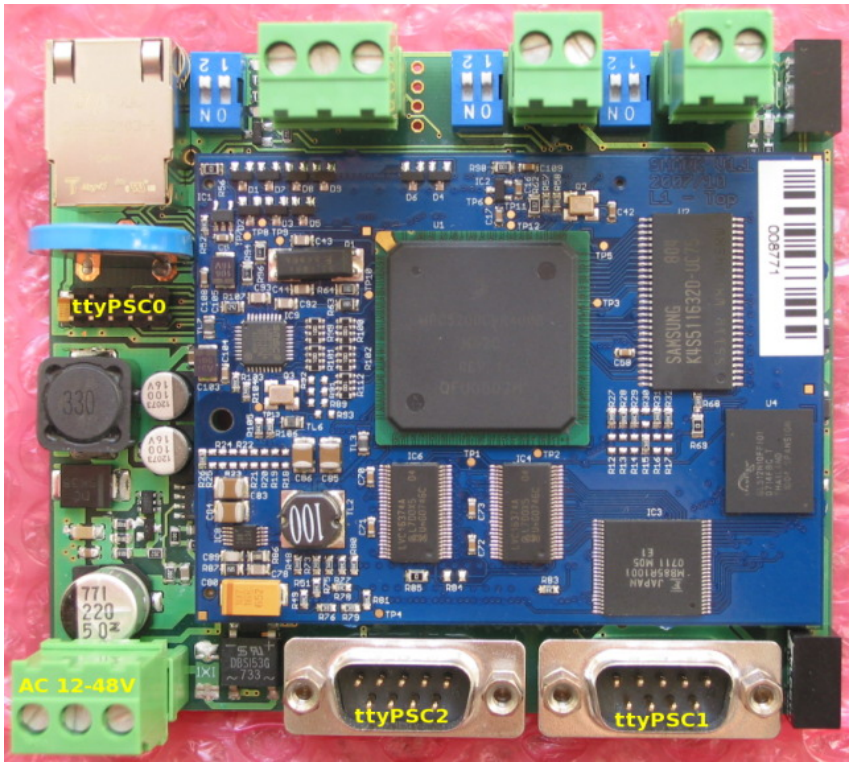
- recvmmsg 7% faster
- sendmmsg 12% faster

Embedded system (MPC5200B)



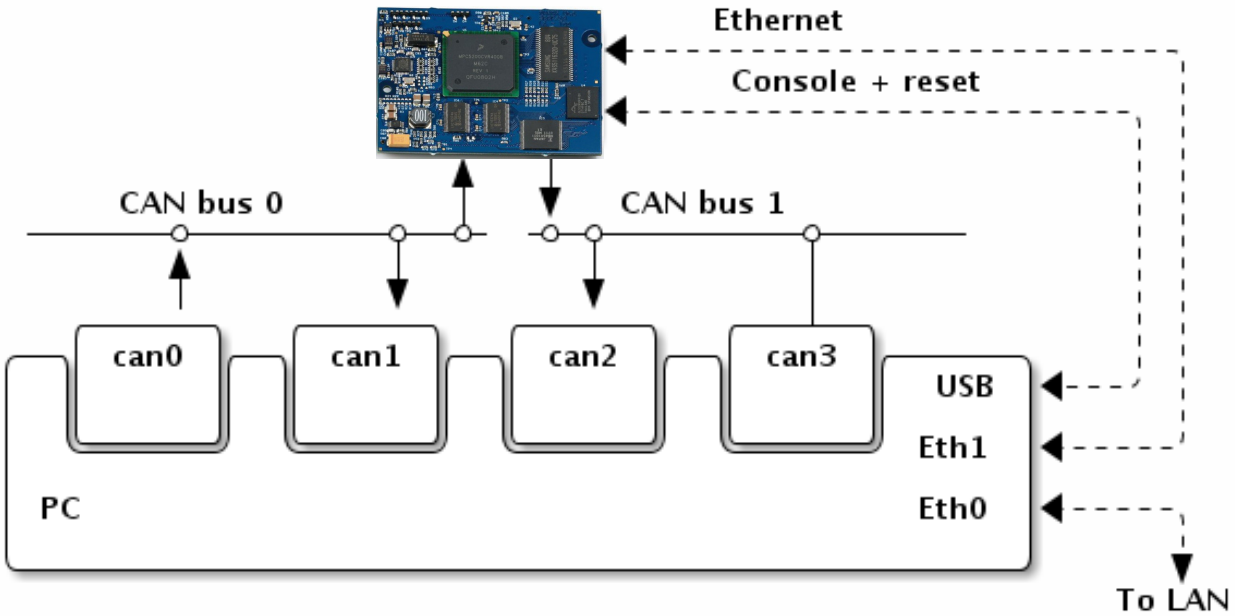
- recvmmsg 19% faster
- sendmmsg 35% faster

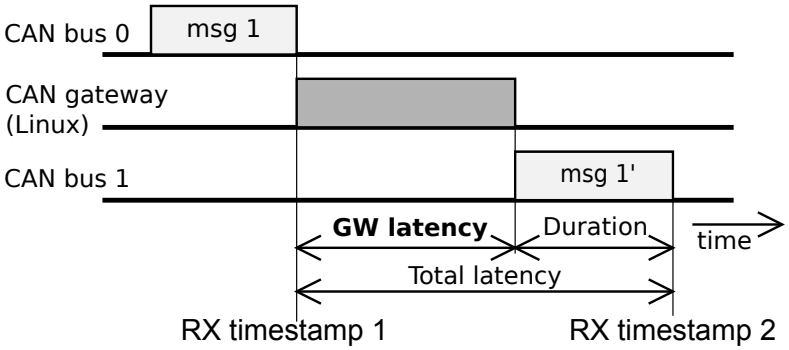
Embedded system used for experiments



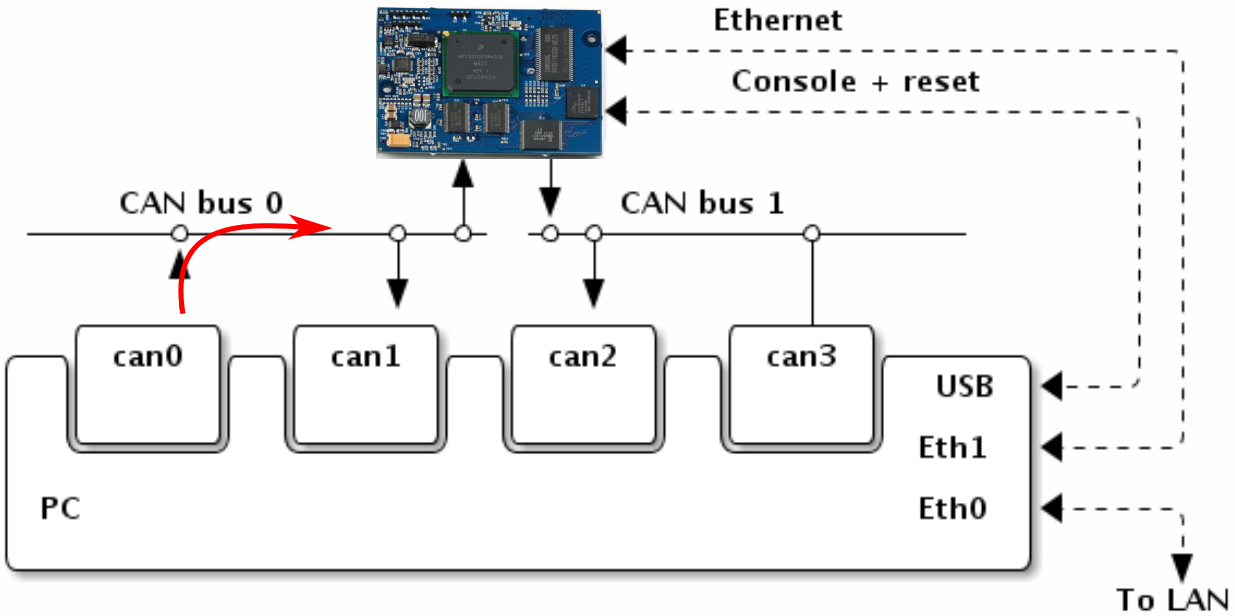
- PowerPC
- Freescale MPC5200B
- 400 Mhz
- 128 MB RAM
- on-chip CAN controller
- Linux 3.12.3

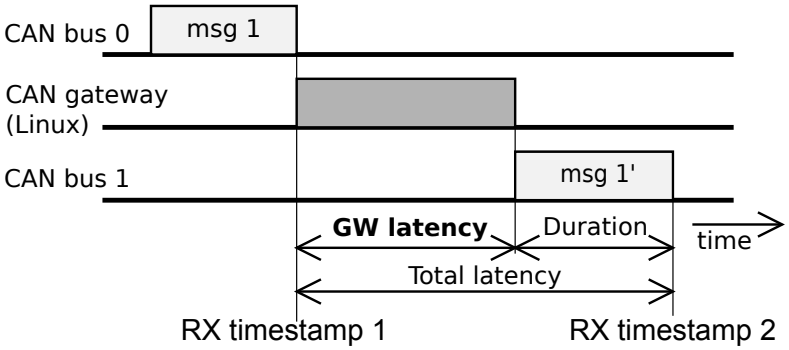
Gateway-based experiments



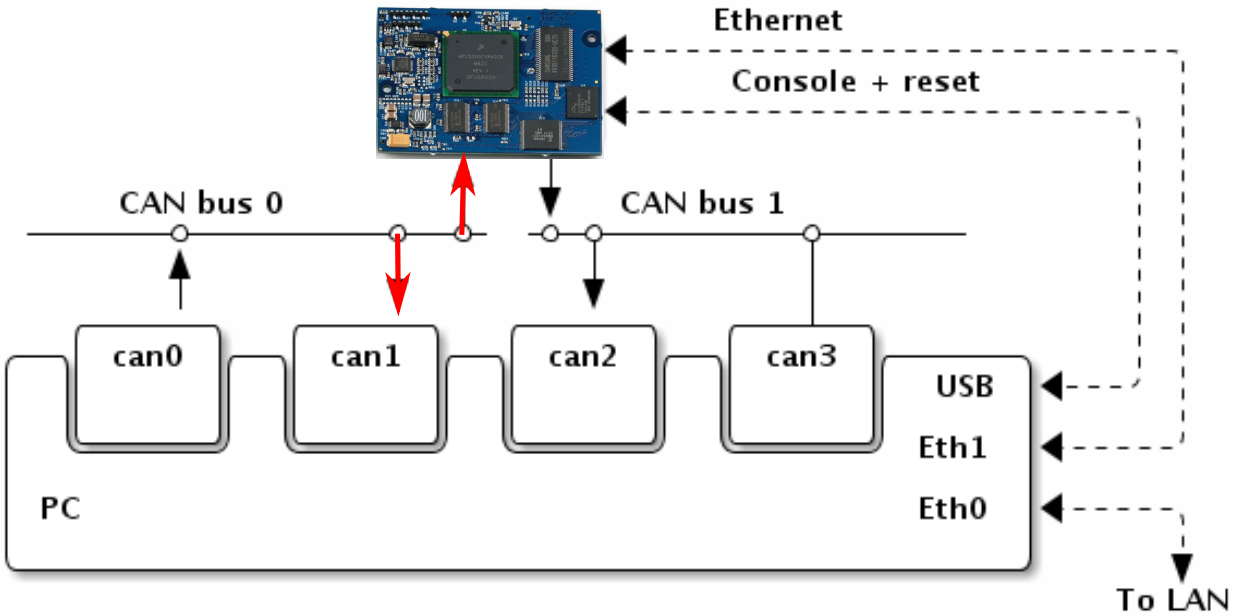


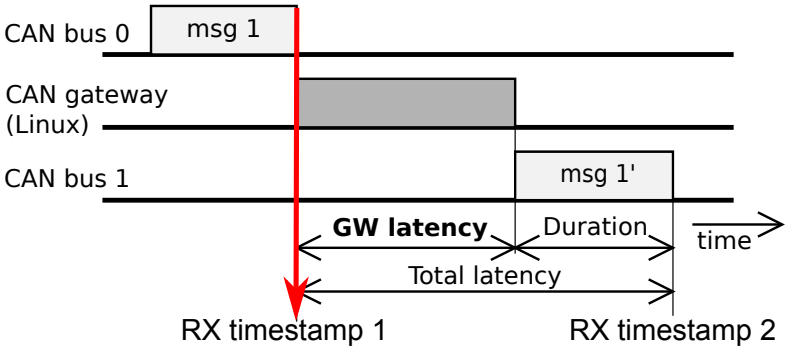
Gateway-based experiments



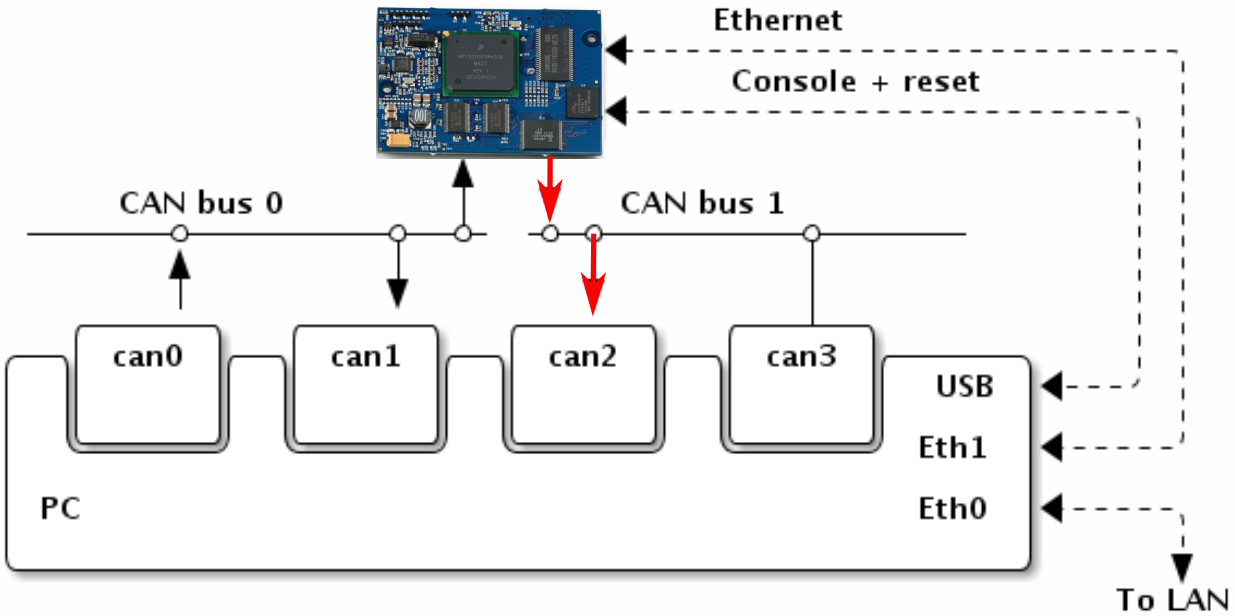


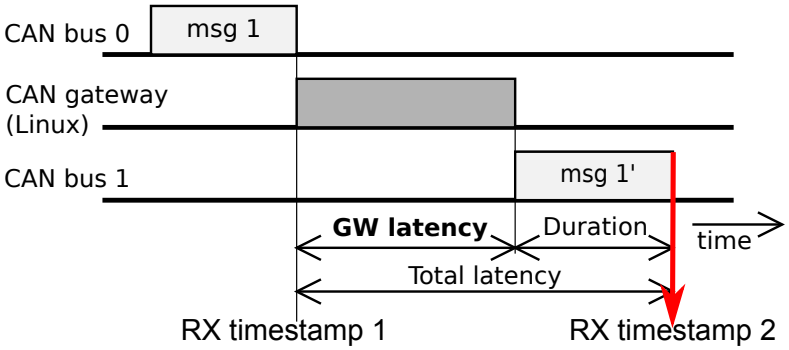
Gateway-based experiments



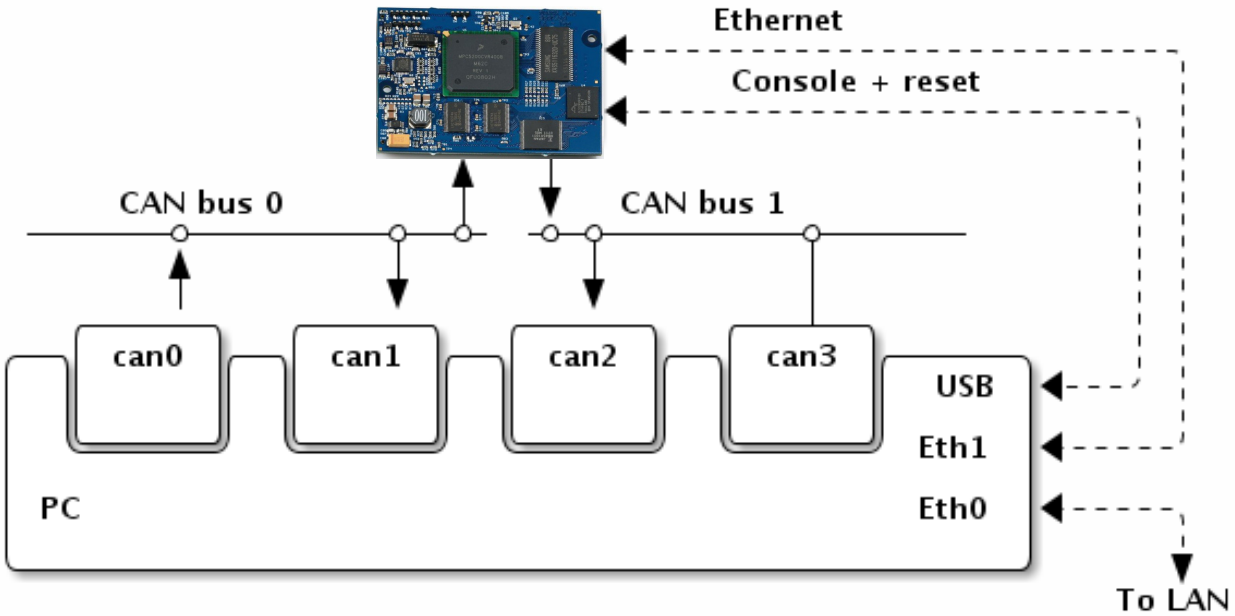


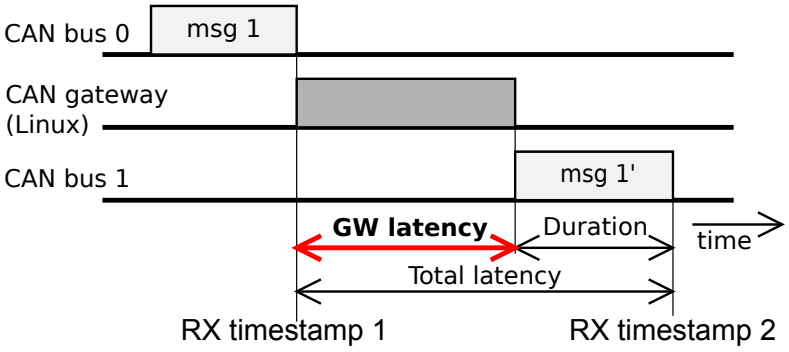
Gateway-based experiments



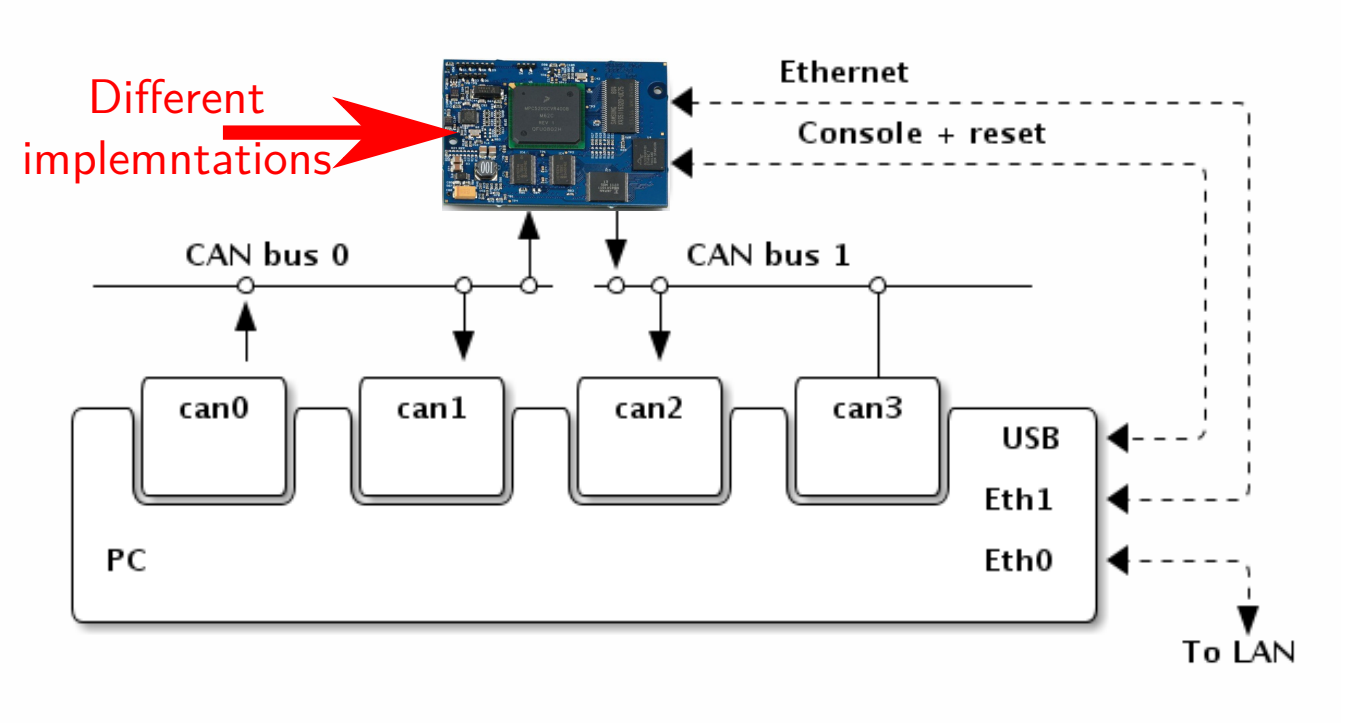


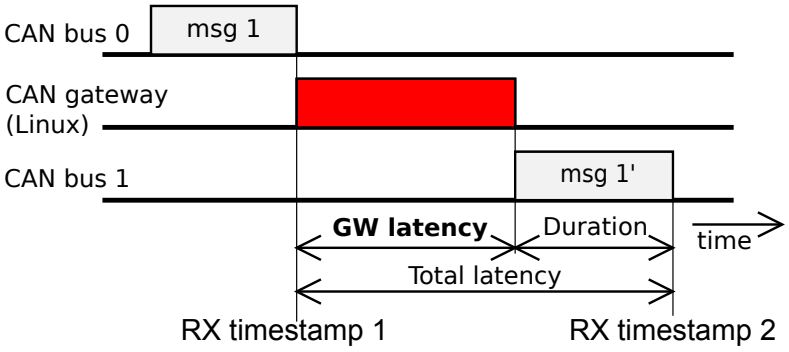
Gateway-based experiments



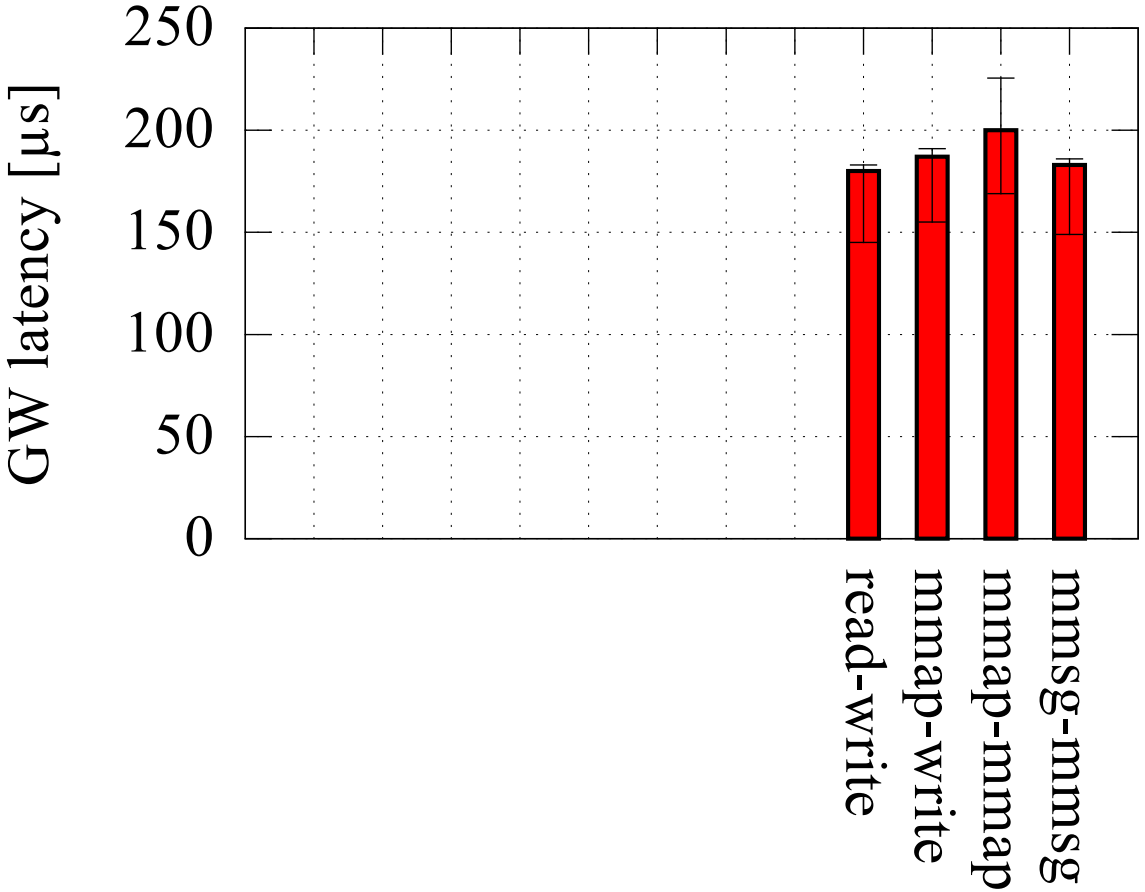


Gateway-based experiments



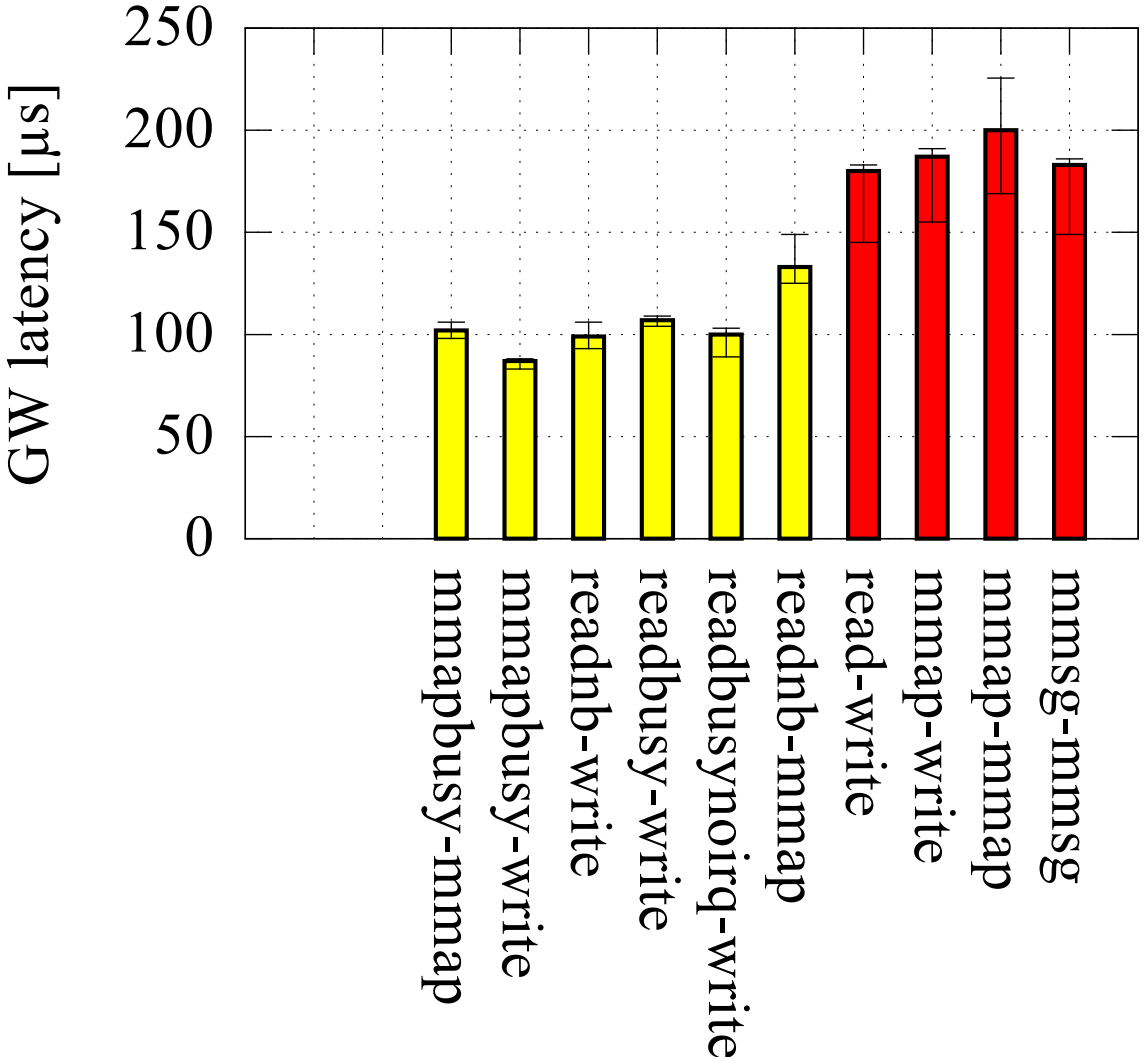


Results – one message at a time



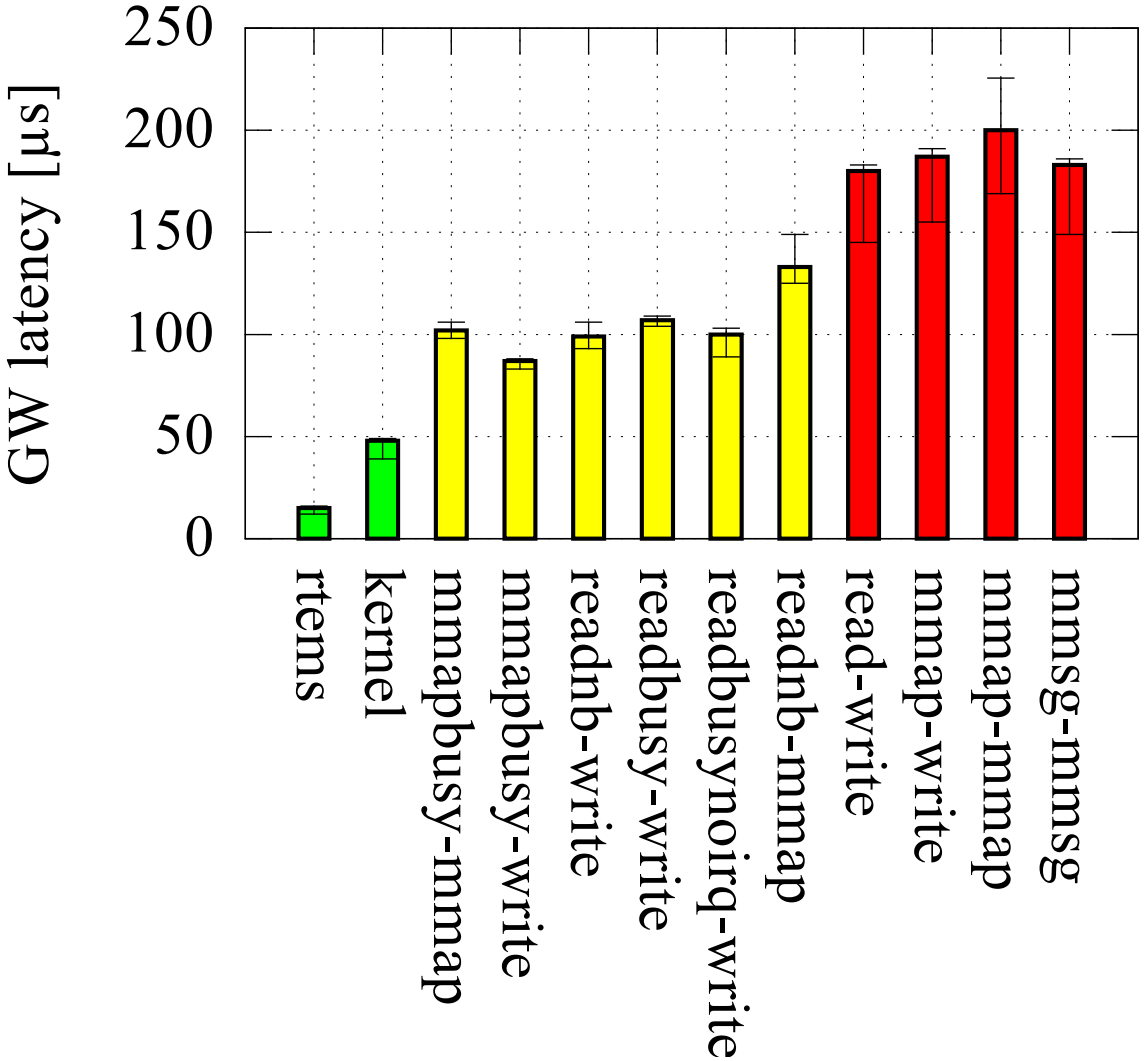
- Blocking system calls
- Non-blocking/busy waiting
- Linux kernel & RTEMS (no Linux)

Results – one message at a time



- Blocking system calls
- Non-blocking/busy waiting
- Linux kernel & RTEMS (no Linux)

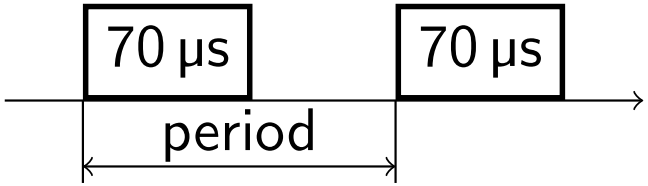
Results – one message at a time

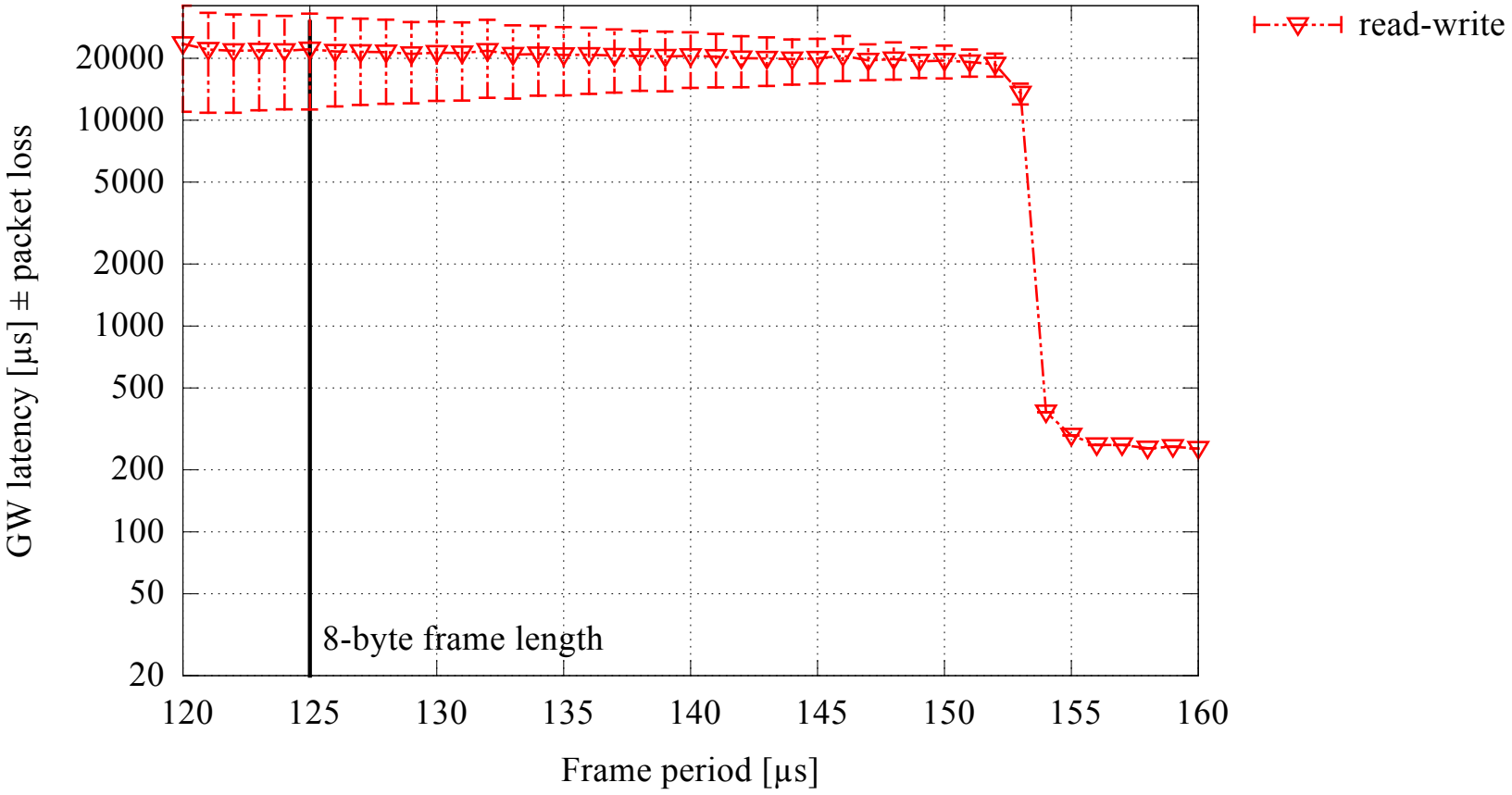


- Blocking system calls
- Non-blocking/busy waiting
- Linux kernel & RTEMS (no Linux)

Results – periodic messages

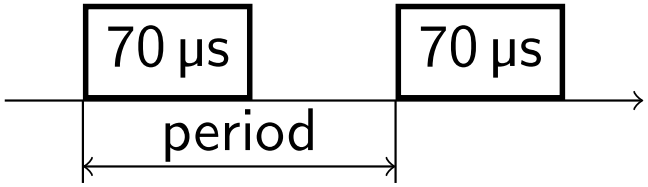
Bitrate: 1 Mbps, Payload: 2 bytes
Median of 3200 frames

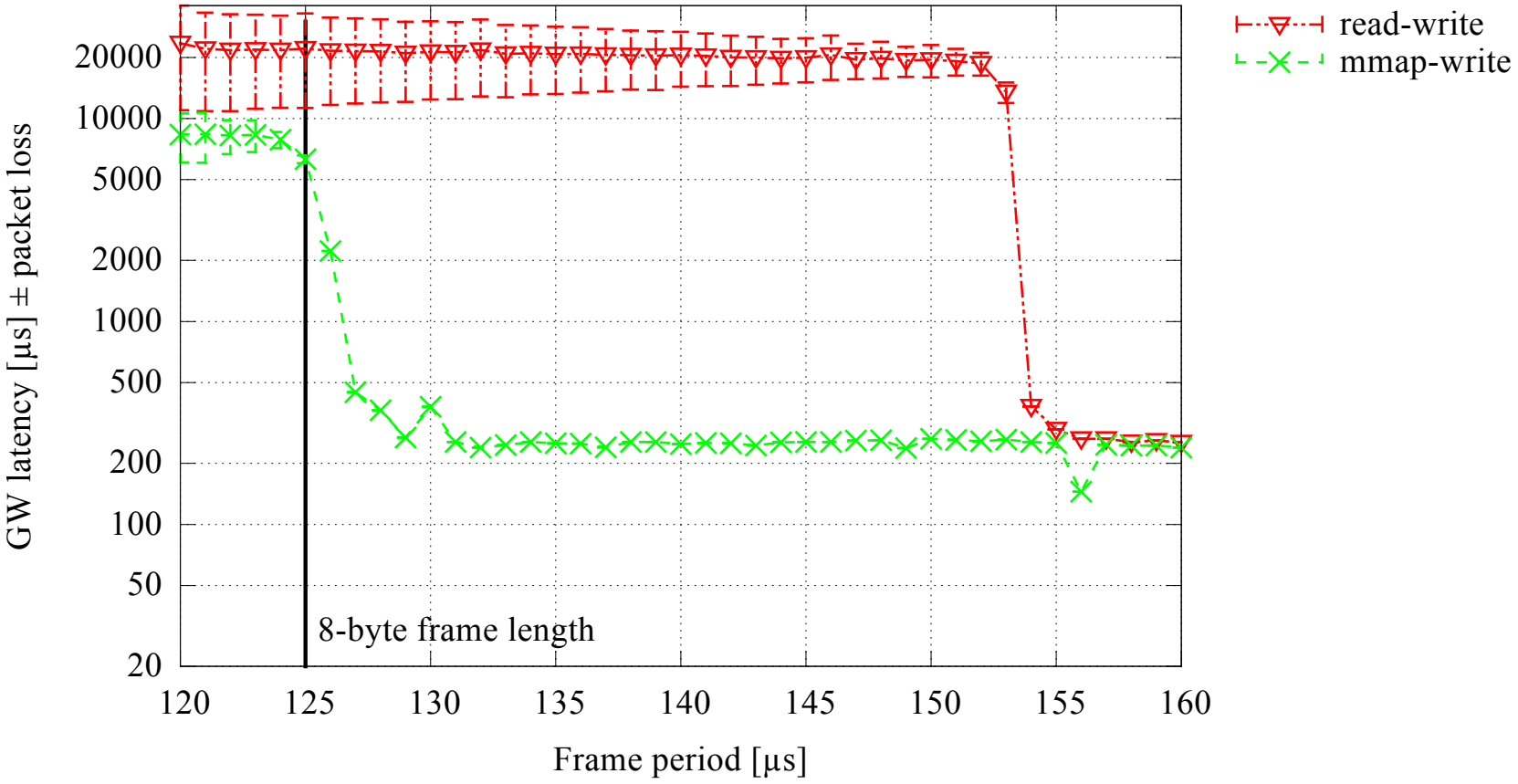




Results – periodic messages

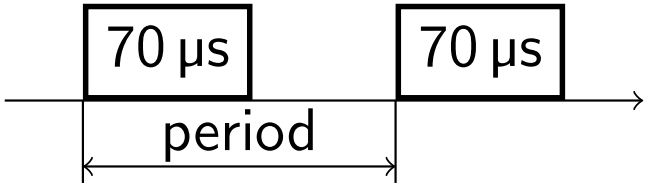
Bitrate: 1 Mbps, Payload: 2 bytes
Median of 3200 frames

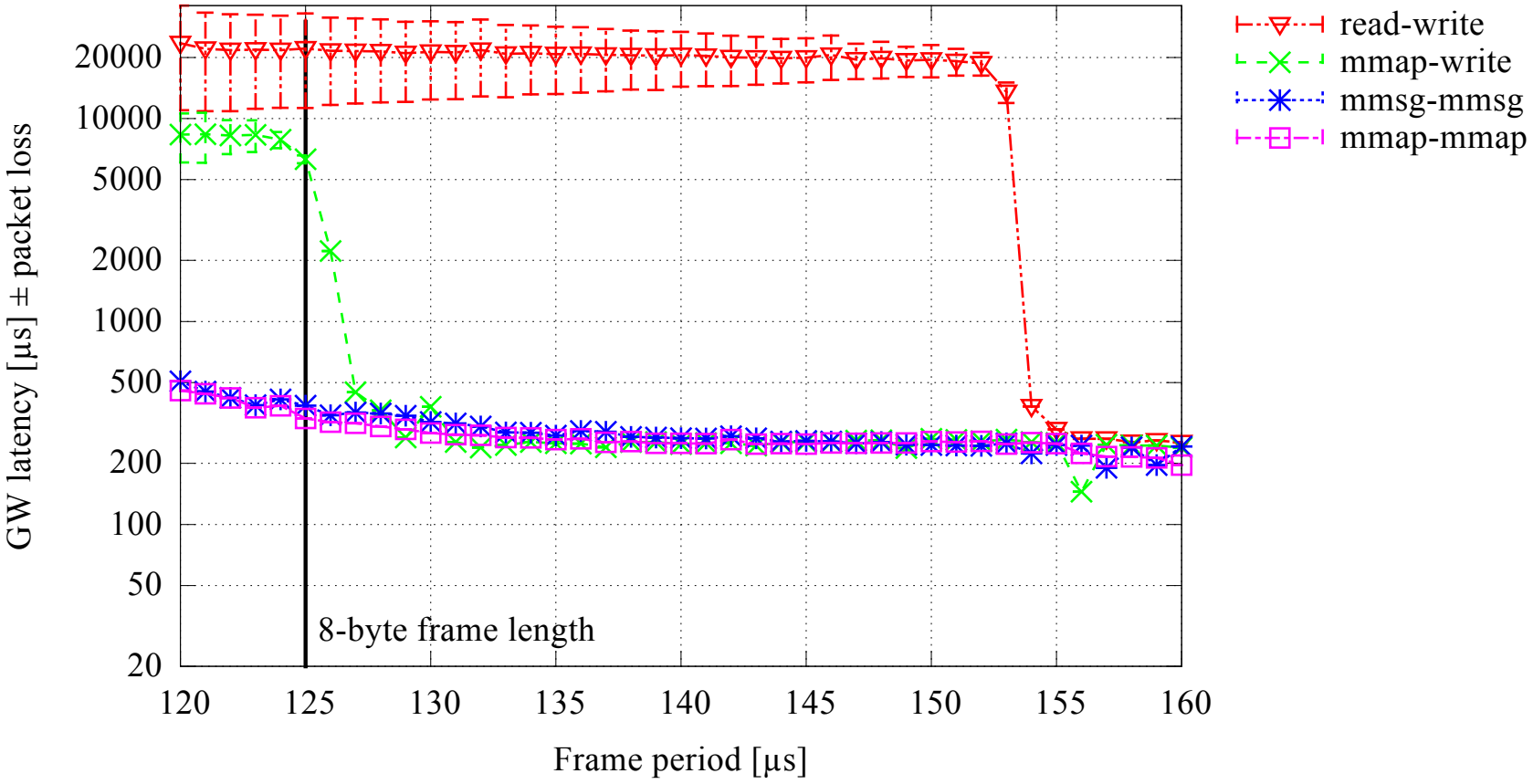




Results – periodic messages

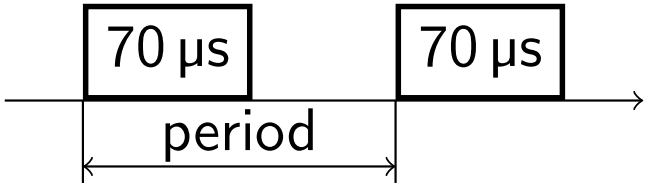
Bitrate: 1 Mbps, Payload: 2 bytes
Median of 3200 frames

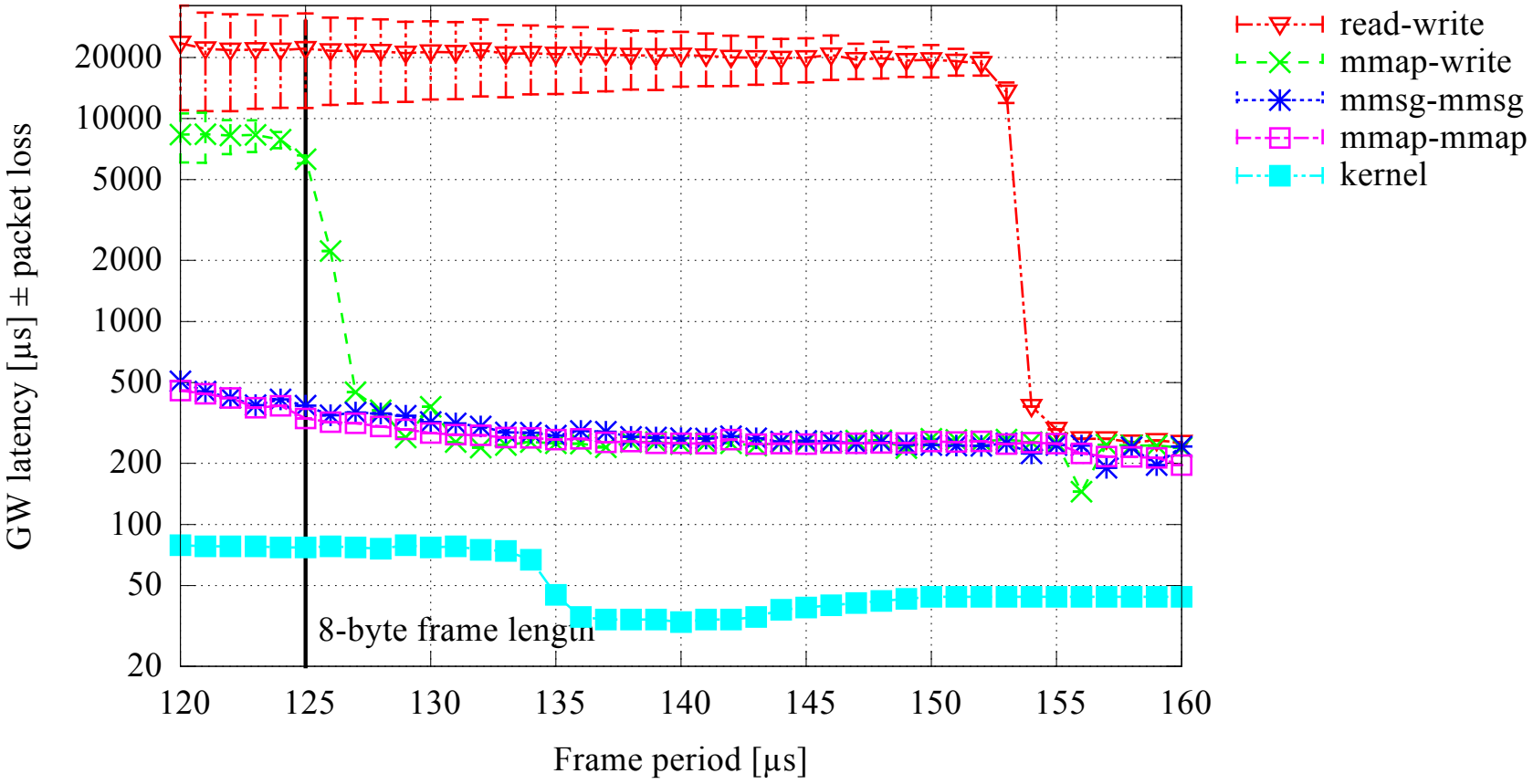




Results – periodic messages

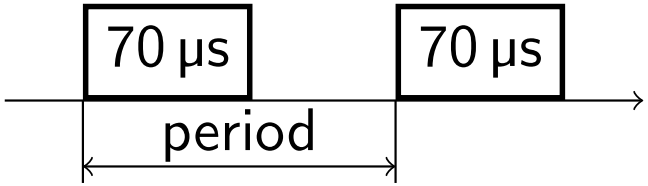
Bitrate: 1 Mbps, Payload: 2 bytes
Median of 3200 frames

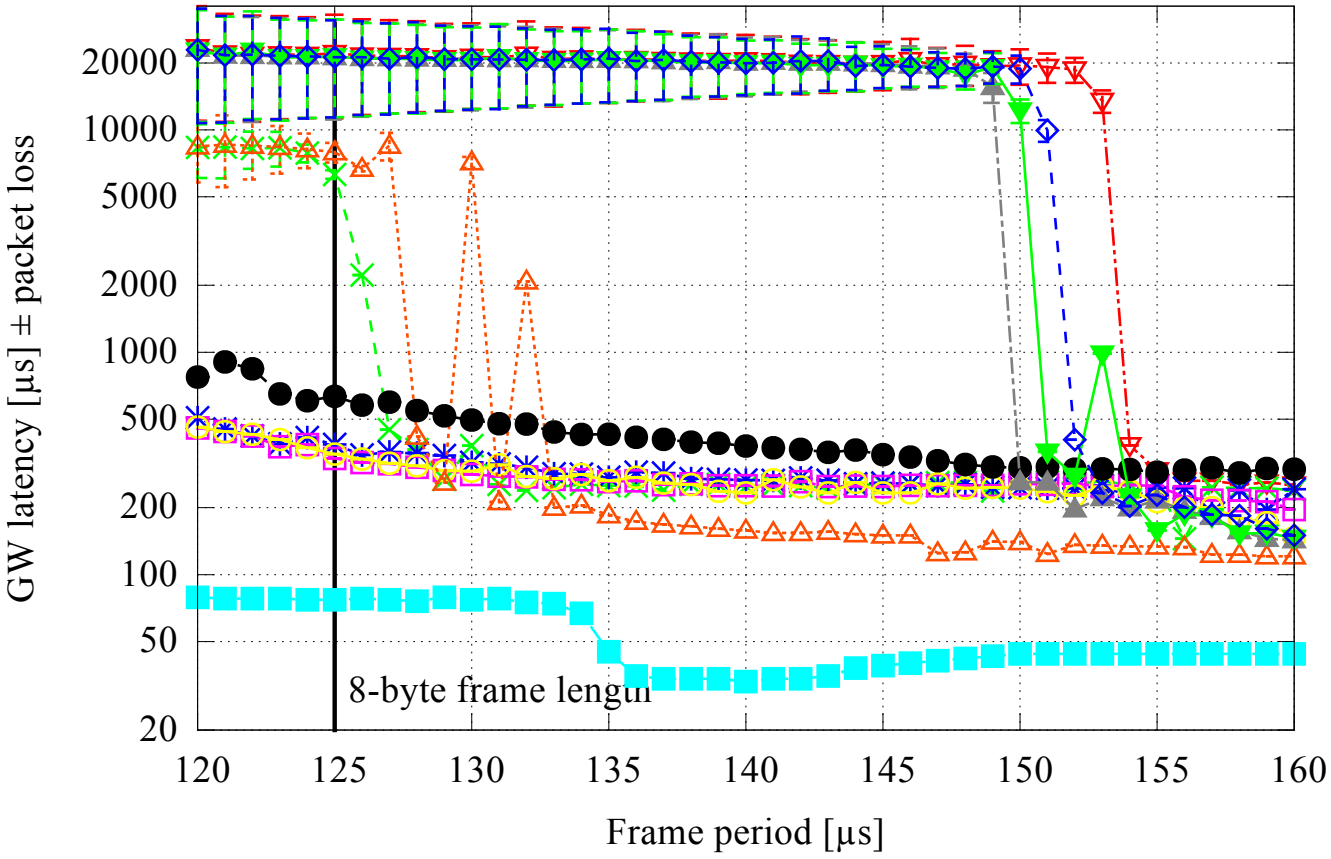




Results – periodic messages

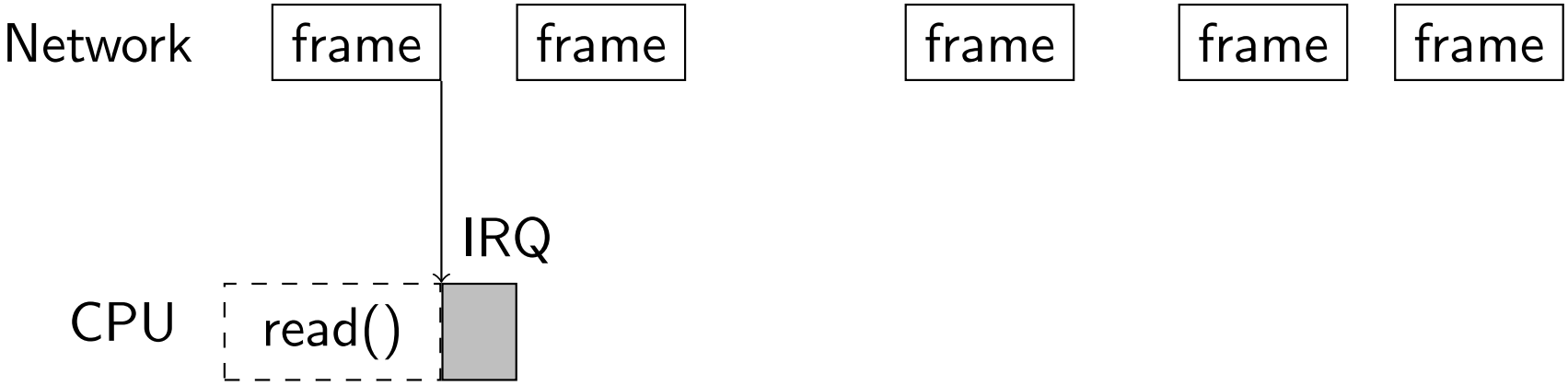
Bitrate: 1 Mbps, Payload: 2 bytes
Median of 3200 frames





Low-latency sockets

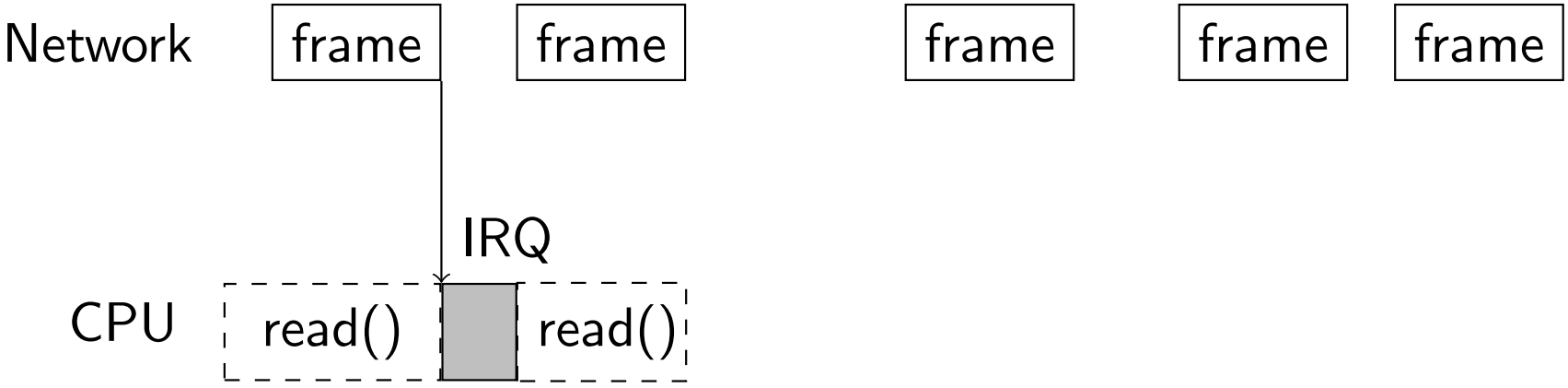
aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Low-latency sockets

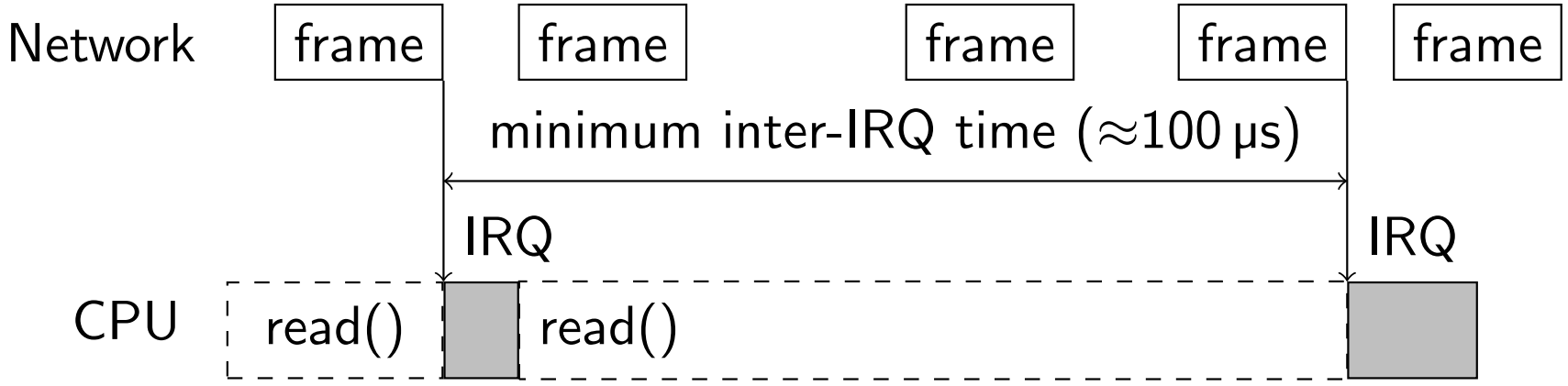
aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Low-latency sockets

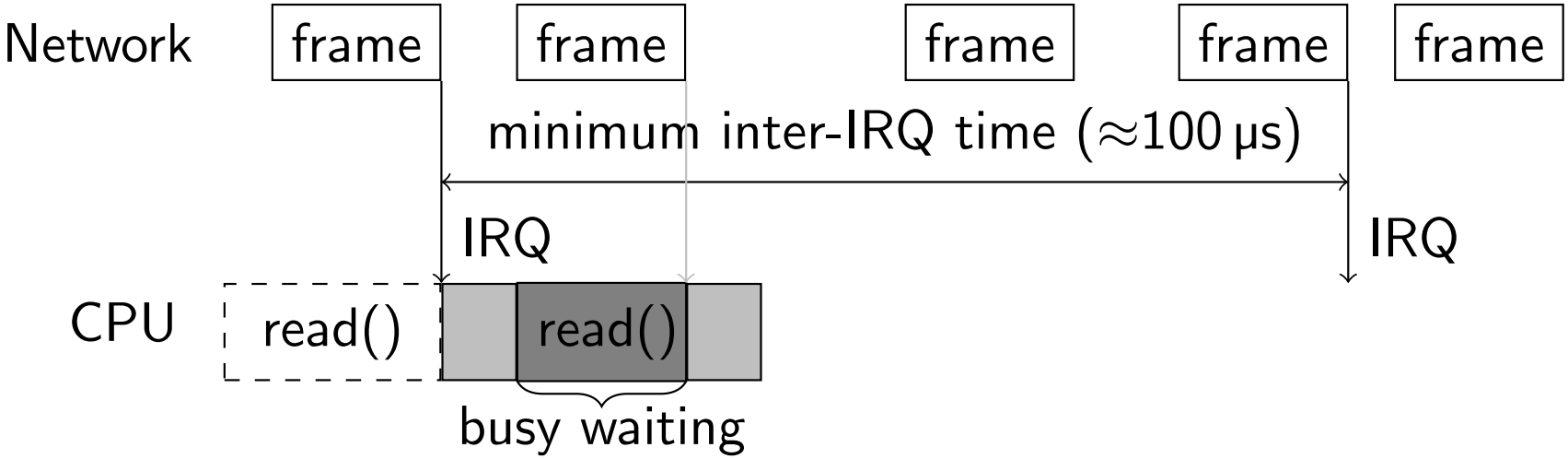
aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Low-latency sockets

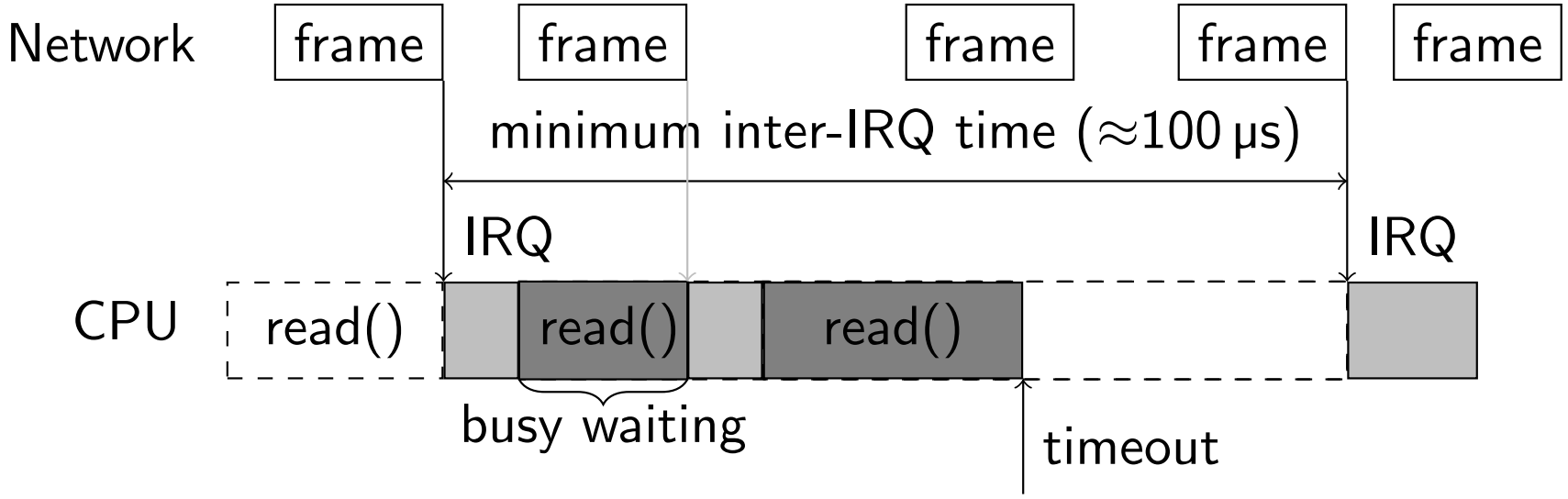
aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Low-latency sockets

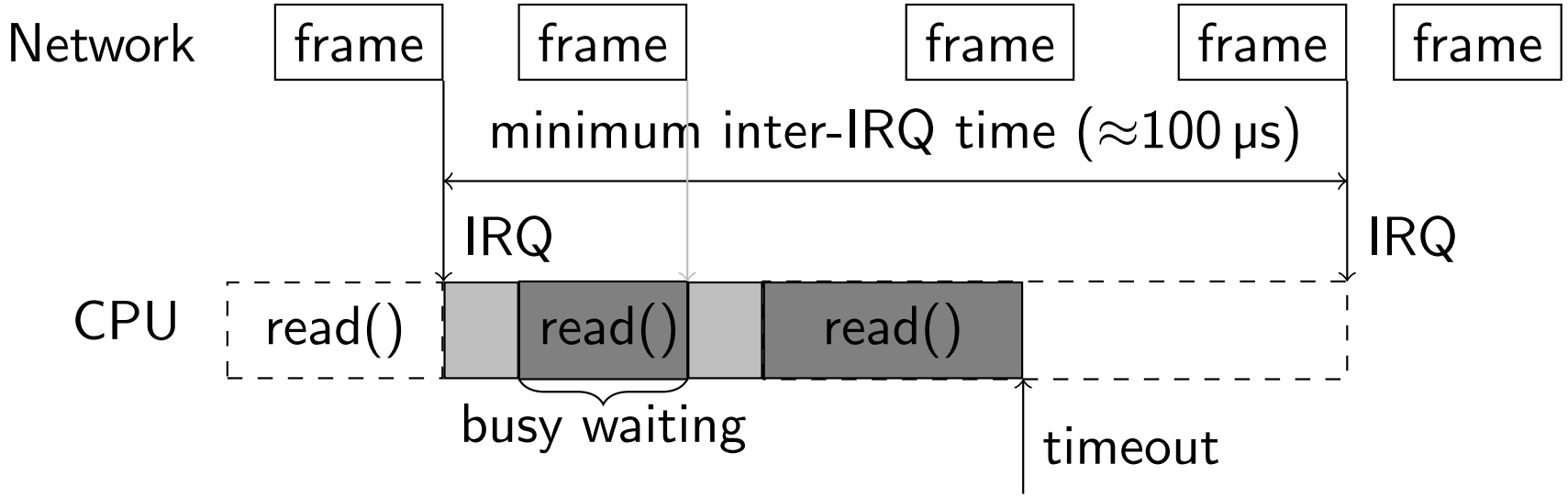
aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Low-latency sockets

aka Busy polling sockets



- Not many CAN controllers offer interrupt coalescing.
- Avoiding IRQ overhead brings only negligible benefits.
- Busy polling is not useful for CAN.

Conclusion

- Experimental performance analysis of Linux network-related user space APIs in the context of CAN bus.
- Interfaces that can handle multiple messages per system call (`recvmmsg()`, memory-mapped sockets) are superior to single-message system calls (`read()`).
  - `recvmmsg()` is simpler to use than memory-mapped sockets.
- New Low-latency sockets are not useful for CAN applications.
- Task wake-up latency.

Conclusion

- Experimental performance analysis of Linux network-related user space APIs in the context of CAN bus.
- Interfaces that can handle multiple messages per system call (`recvmmsg()`, memory-mapped sockets) are superior to single-message system calls (`read()`).
  - `recvmmsg()` is simpler to use than memory-mapped sockets.
- New Low-latency sockets are not useful for CAN applications.
- Task wake-up latency.

Thank you!