Current status of Linux real-time on its way to mainline, Theoretical part:

Rationale of turning Linux into an RTOS and challenges to master it

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

Why real-time?

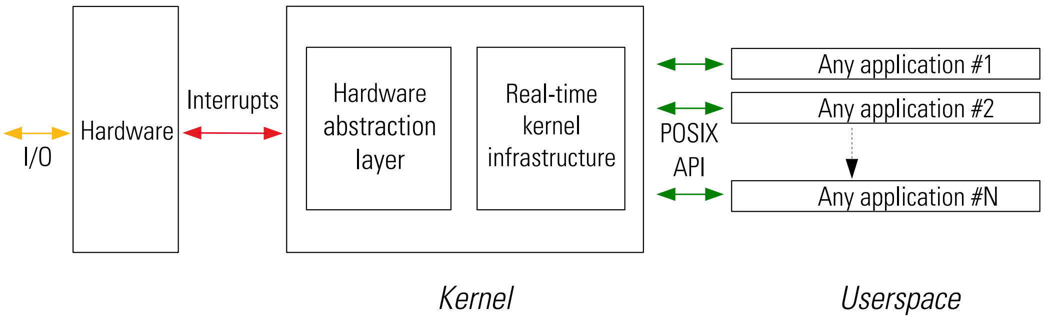
- About 20 years ago:
  - Engineers who wanted to use Linux for embedded industrial devices were asked what prevented them from doing so.
    - About 30% said: Because there was no real-time
    - About 10% said: Because there was no safety certification

Why real-time?

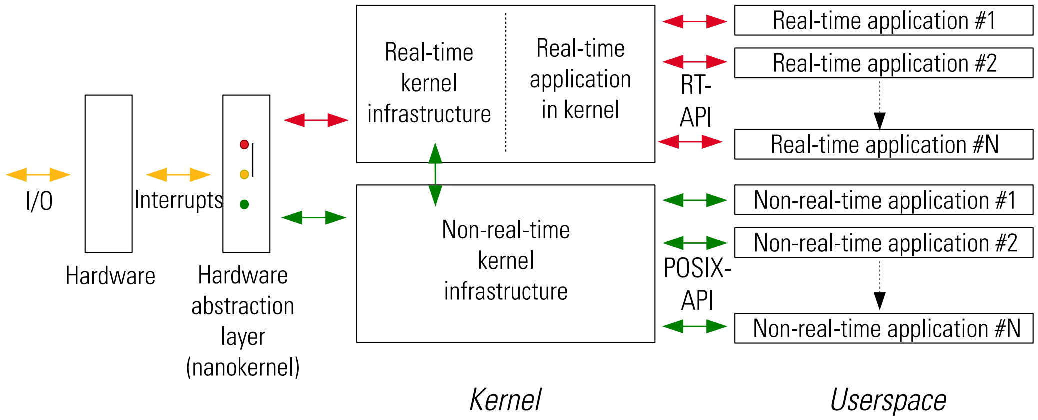
- About 20 years ago:
  - Engineers who wanted to use Linux for embedded industrial devices were asked what prevented them from doing so.
    - **About 30% said: Because there was no real-time**
    - About 10% said: Because there was no safety certification

Which real-time?

"Single-kernel approach"



"Dual-kernel approach"



Which real-time?

"Single-kernel approach"	"Dual-kernel approach"
• Reuse of drivers	• Mostly new drivers
• More elegant	• More like a "hack"
• Technically very challenging	• Probably less challenging
• Standard Linux expertise required	• Expertise beyond Linux required
• Hope for mainline merge	• No hope for mainline merge
• Long way to go for mainline	• No way to go for mainline
• No upgrade changes when mainline	• Upgrade changes always required

?

Challenges of the single-kernel approach

- Mainline kernel lacks a mandatory or nice-to-have feature for RT
  - Convince mainline kernel developers to accept the feature
- Mainline kernel has an implementation that blocks an RT feature
  - Convince mainline kernel developers to re-implement it
- Mainline kernel needs an adaptation to improve RT
  - Convince mainline kernel developers to accept the adaptation
- Mainline kernel suffers a regression after making it RT
  - Communicate the regression and find a mitigation

Mainline kernel lacks a mandatory feature for RT

Examples

- Priority-inheritance mutexes
- High-resolution timer
- Forced IRQ threading

Mainline kernel lacks a mandatory feature for RT

Examples

- Priority-inheritance mutexes
  - Also beneficial for mainline
- High-resolution timer
- Forced IRQ threading

Mainline kernel lacks a mandatory feature for RT

Examples

- Priority-inheritance mutexes
  - Also beneficial for mainline
- High-resolution timer
  - Connected to dynamic tick to enable energy efficiency
- Forced IRQ threading

Mainline kernel lacks a mandatory feature for RT

Examples

- Priority-inheritance mutexes
  - Also beneficial for mainline
- High-resolution timer
  - Connected to dynamic tick to enable energy efficiency
- Forced IRQ threading
  - Allows easier trouble shooting of IRQ handlers

Mainline kernel has an implementation that blocks RT

Example "softIRQ split"

Initial scenario:

In Kernel 3.6 the so-called "softIRQ split" was merged to mainline. This made it possible to immediately forward the priority of the IRQ kernel thread to the softIRQ handler. For example, only two actions were required for the transmission of network packets in real-time:

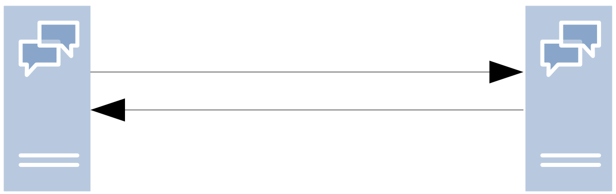
- Set the priority of the IRQ thread of the network adapter to a suitable value
- Set the priority of the related user-space application to a suitable value

(see "Peer-to-peer UDP duplex link" at the OSADL Webpage https://www.osadl.org/?id=930)

Peer-to-peer UDP duplex link for real-time network

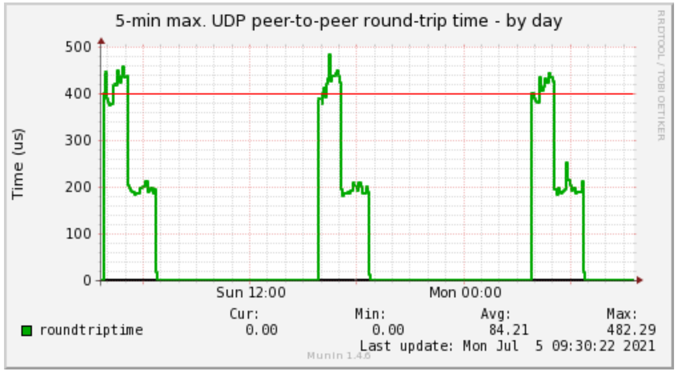
Client at Rack #5, slot #4

Server at Rack #1, slot #2

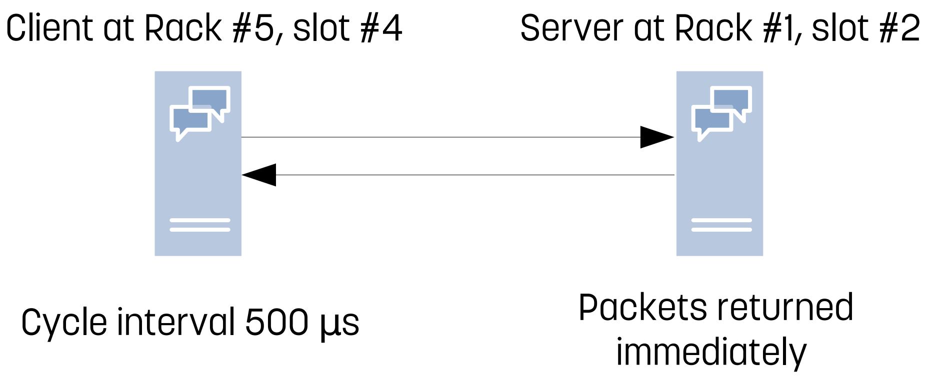


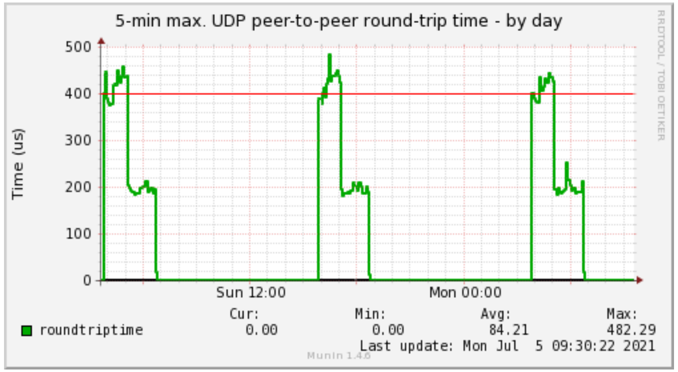
Cycle interval 500 µs

Packets returned immediately



Peer-to-peer UDP duplex link for real-time network





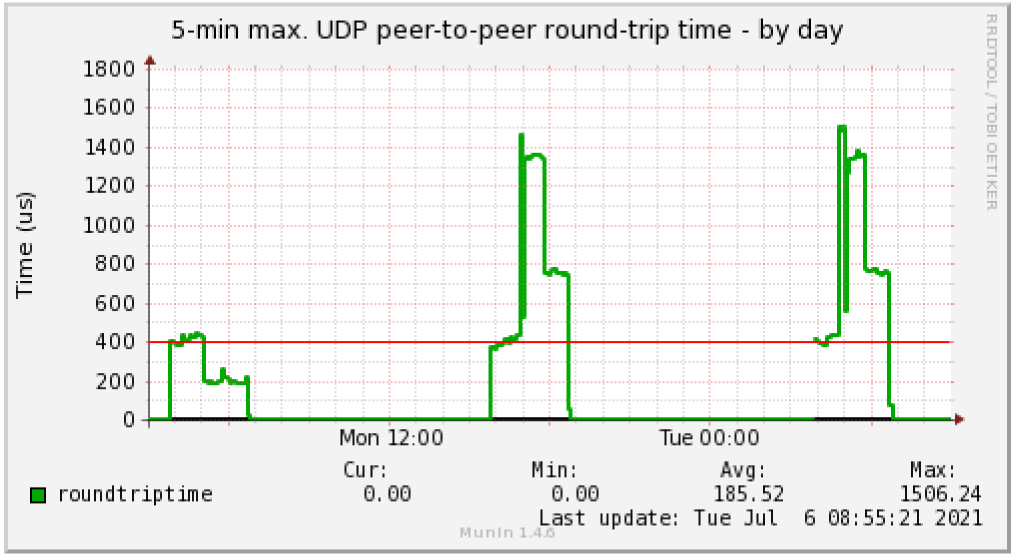
Real-time and normal traffic could even be run simultaneously without interference:

- Use different VLAN channels with different priorities, e.g. 7 for RT, 0 for normal traffic
- Configure *ingress* and *egress* mapping to connect network with socket layer priority

Mainline kernel has an implementation that blocks RT

Rework of the softIRQ framework between kernel 4.14 and 4.16

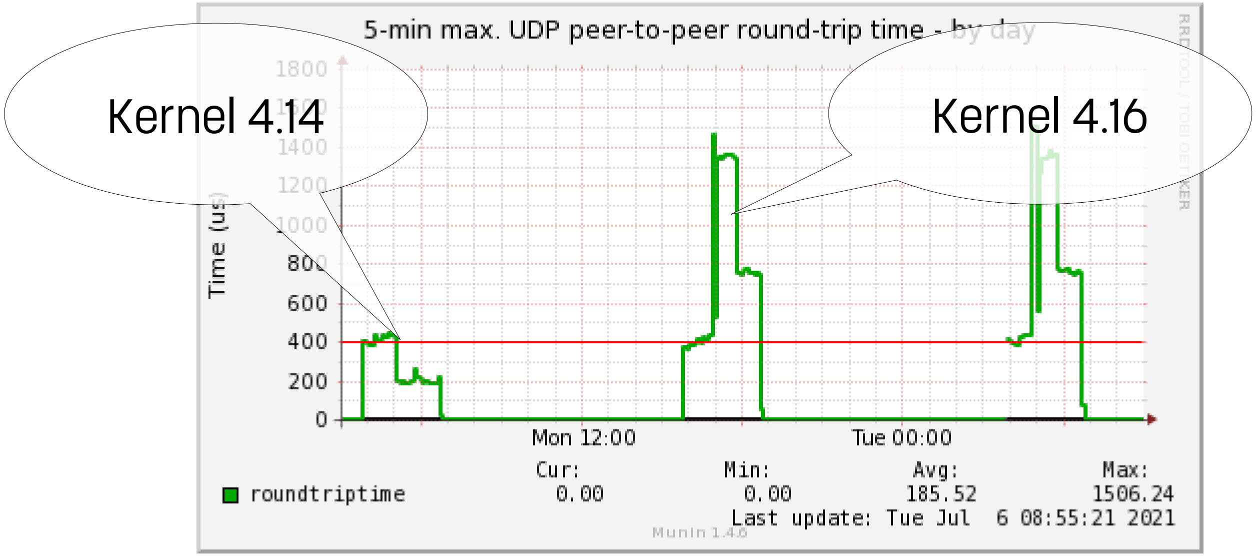
As a consequence of the softIRQ rework, the "softIRQ split" could no longer be used. Therefore, large latency values occur after upgrading from 4.14 to 4.16:



Mainline kernel has an implementation that blocks RT

Rework of the softIRQ framework between kernel 4.14 and 4.16

As a consequence of the softIRQ rework, the "softIRQ split" could no longer be used. Therefore, large latency values occur after upgrading from 4.14 to 4.16:



Mainline kernel has an implementation that blocks RT

Rework of the softIRQ framework between kernel 4.14 and 4.16

As a work-around, a multi-core processor may be used and a core be isolated to exclusively handle the network interrupt. Since there is a dedicated softIRQ handler per core, interference with other IRQ handlers may be avoided.

Mainline kernel has an implementation that blocks RT

Rework of the softIRQ framework between kernel 4.14 and 4.16

As a work-around, a multi-core processor may be used and a core be isolated to exclusively handle the network interrupt. Since there is a dedicated softIRQ handler per core, interference with other IRQ handlers may be avoided.

OSADL is working on this issue and will update the information on the related Web page accordingly. For the time being, the original performance could not yet be established.

Conclusion

- In many cases, mainline kernel developers could be convinced to accept merge requests of RT features, since these features were also beneficial to the non-RT kernel – or at least, it was possible to claim that they were.

Conclusion

- In many cases, mainline kernel developers could be convinced to accept merge requests of RT features, since these features were also beneficial to the non-RT kernel – or at least, it was possible to claim that they were.
- In the case of implementations that would have to be removed or reverted, it was less easy – if at all – to convince kernel developers to sacrifice the implementation because of RT. There is still some convincing to be done here.