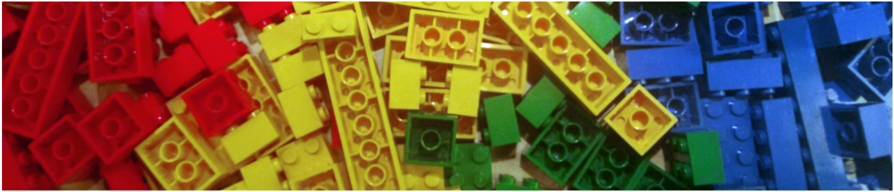


Analysis and Comparison of Embedded Network Stacks

Design and Evaluation of the GNRC Network Stack

Martine Lenders (4206090, mlenders@inf.fu-berlin.de)

Master thesis defense

Freie Universität Berlin, Department for Computer Science

Supervisors: Dr. Emmanuel Baccelli, Univ.-Prof. Dr. Jochen Schiller

2016-06-27

Outline

1. Introduction
2. RIOT
3. GNRC
4. Evaluation of GNRC
5. Conclusion

Outline

1. **Introduction**
2. RIOT
3. GNRC
4. Evaluation of GNRC
5. Conclusion

The Internet of Things

- *"Internet of Things"* = broad, generic term
  - Home automation
  - Industry 4.0
  - Cars
  - Health surveillance
  - Wildlife surveillance
  - Wireless sensor networks
  - ...

The IoT – Constraints & Requirements

- *Large address space*: $>$ 10 Internet connected devices per person
- *Low energy requirements*
  - *Low processing power*: a few MHz
  - *Small memory*: $\leq$ 10 KiB RAM, $\leq$ 100 KiB flash
  - *Lossy transmission medium*: IEEE 802.15.4, Bluetooth Low-Energy, NFC

The IoT – Constraints & Requirements

- *Large address space*: $>$ 10 Internet connected devices per person
- *Low energy requirements*
  - *Low processing power*: a few MHz
  - *Small memory*: $\leq$ 10 KiB RAM, $\leq$ 100 KiB flash
  - *Lossy transmission medium*: IEEE 802.15.4, Bluetooth Low-Energy, NFC
- Constraints govern need for:
  - specific OSs: TinyOS, Contiki, FreeRTOS, **RIOT**
  - specific communication protocols: ZigBee, Z-Wave, **IETF's IPv6-based IoT suite**

Approaching a solution

Problem 1 *Large address space*

Approaching a solution

Problem 1 *Large address space* ⇒ IPv4 unsuitable

$2^{32} \approx 4.3 \cdot 10^9$possible addresses $\ll 7.4 \cdot 10^{10}$devices

⇒ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Approaching a solution

Problem 1 *Large address space* $\Rightarrow$ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^{9} \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

$\Rightarrow$ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$?

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs. 1280 B *minimum* MTU in IPv6 (header alone 40 B)

Approaching a solution

Problem 1 *Large address space* $\Rightarrow$ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^9 \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

$\Rightarrow$ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs. 1280 B *minimum* MTU in IPv6 (header alone 40 B)
$\Rightarrow$ Low-level fragmentation + header compression
AKA **6LoWPAN**

Approaching a solution

Problem 1 *Large address space* $\Rightarrow$ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^{9} \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

$\Rightarrow$ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs.
1280 B *minimum* MTU in IPv6 (header alone 40 B)
$\Rightarrow$ Low-level fragmentation + header compression
AKA **6LoWPAN**

Problem 3 TCP too complex for *small memory*
Congestion control harmful on *lossy transmission medium*

Approaching a solution

Problem 1 *Large address space* $\Rightarrow$ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^9 \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

$\Rightarrow$ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs.
1280 B *minimum* MTU in IPv6 (header alone 40 B)
$\Rightarrow$ Low-level fragmentation + header compression
AKA **6LoWPAN**

Problem 3 TCP too complex for *small memory*
Congestion control harmful on *lossy transmission medium*
$\Rightarrow$ **UDP** instead

Approaching a solution

Problem 1 *Large address space* $\Rightarrow$ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^{9} \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

$\Rightarrow$ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs.
1280 B *minimum* MTU in IPv6 (header alone 40 B)
$\Rightarrow$ Low-level fragmentation + header compression
AKA **6LoWPAN**

Problem 3 TCP too complex for *small memory*
Congestion control harmful on *lossy transmission medium*
$\Rightarrow$ **UDP** instead

Problem 4 No TCP, no HTTP, no WWW?

Approaching a solution

Problem 1 *Large address space* ⇒ IPv4 unsuitable

$$2^{32} \approx 4.3 \cdot 10^9 \text{possible addresses} \ll 7.4 \cdot 10^{10} \text{devices}$$

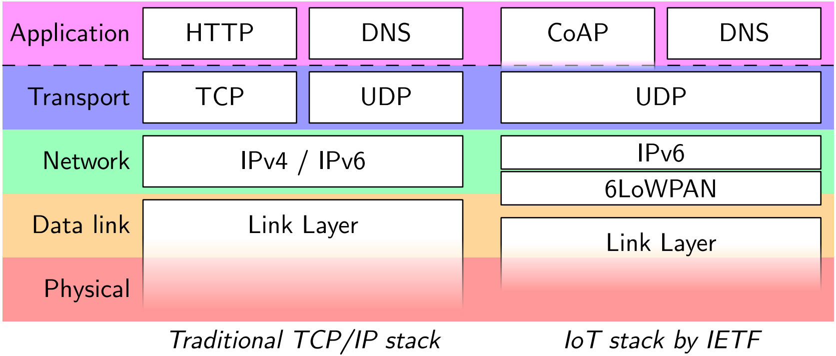
⇒ **IPv6** $(2^{128} \approx 3.4 \cdot 10^{38})$

Problem 2 e.g. IEEE 802.15.4 frame size max. 127 B vs.
1280 B *minimum* MTU in IPv6 (header alone 40 B)
⇒ Low-level fragmentation + header compression
AKA **6LoWPAN**

Problem 3 TCP too complex for *small memory*
Congestion control harmful on *lossy transmission medium*
⇒ **UDP** instead

Problem 4 No TCP, no HTTP, no WWW?
⇒Non-TCP alternative **CoAP**

Summary: The IETF protocol suite



Existing solutions

existing stack (RIOT)

(+) IoT support

Existing solutions

existing stack (RIOT)

- (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering
- (–) no clear structure / unmaintainable

Existing solutions

existing stack (RIOT)

- (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering
- (–) no clear structure / unmaintainable

BLIP (TinyOS) (+) Small memory footprint (+) IoT support

Existing solutions

existing stack (RIOT)
- (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering
- (–) no clear structure / unmaintainable

BLIP (TinyOS)
- (+) Small memory footprint (+) IoT support
- (–) Exotic C-dialect nesC

Existing solutions

existing stack (RIOT)

- (+) IoT support
- (−) Very rigid in selection of protocols
- (−) Single-packet buffering
- (−) no clear structure / unmaintainable

BLIP (TinyOS) (+) Small memory footprint (+) IoT support

- (−) Exotic C-dialect nesC

uIP (Contiki) (+) Well established operating system (+) IoT support

Existing solutions

existing stack (RIOT)
- (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering
- (–) no clear structure / unmaintainable

BLIP (TinyOS)
- (+) Small memory footprint (+) IoT support
- (–) Exotic C-dialect nesC

uIP (Contiki)
- (+) Well established operating system (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering

Existing solutions

existing stack (RIOT)
- (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering
- (–) no clear structure / unmaintainable

BLIP (TinyOS)
- (+) Small memory footprint (+) IoT support
- (–) Exotic C-dialect nesC

uIP (Contiki)
- (+) Well established operating system (+) IoT support
- (–) Very rigid in selection of protocols
- (–) Single-packet buffering

lwIP
- (+) Well established software (+) Modular
- (+) OS independent

Existing solutions

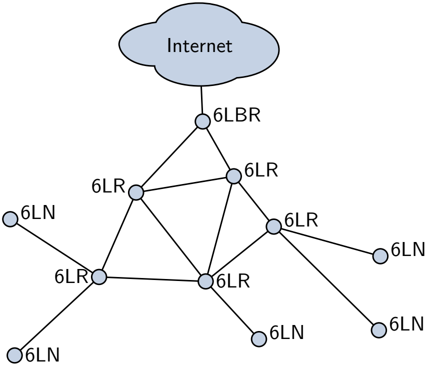
existing stack (RIOT)
- (+) IoT support
- (−) Very rigid in selection of protocols
- (−) Single-packet buffering
- (−) no clear structure / unmaintainable

BLIP (TinyOS) (+) Small memory footprint (+) IoT support
- (−) Exotic C-dialect nesC

uIP (Contiki) (+) Well established operating system (+) IoT support
- (−) Very rigid in selection of protocols
- (−) Single-packet buffering

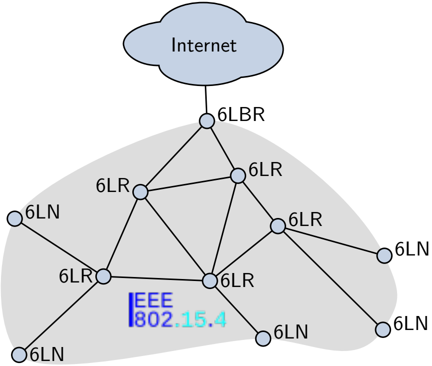
lwIP (+) Well established software (+) Modular
- (+) OS independent
- (−) Tricky to configure
- (−) At start of thesis: no IoT support

Another thing to consider in LoWPANs



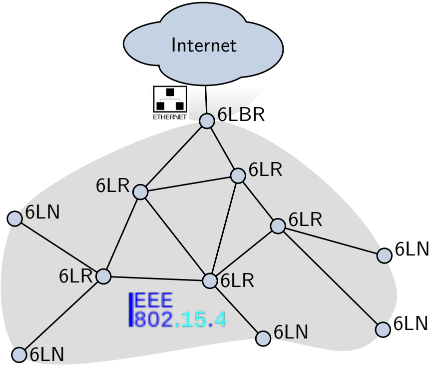
6LBR: border router
6LR: router
6LN: non-routing host

Another thing to consider in LoWPANs



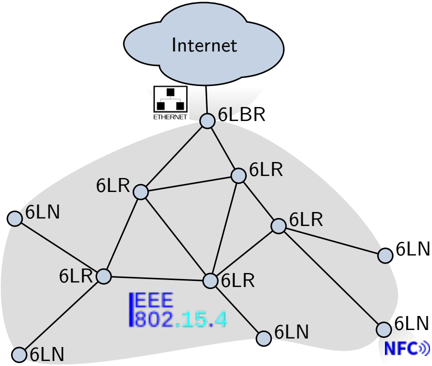
6LBR: border router
6LR: router
6LN: non-routing host

Another thing to consider in LoWPANs



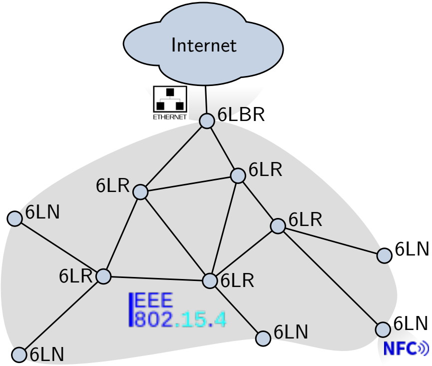
6LBR: border router
6LR: router
6LN: non-routing host

Another thing to consider in LoWPANs



6LBR: border router
6LR: router
6LN: non-routing host

Another thing to consider in LoWPANs



6LBR: border router
6LR: router
6LN: non-routing host

⇒ Multi-interface support required (only BLIP and lwIP provides that)

Requirements

Functional Requirements:

- Focus on IoT protocols
- Multiple interface support
- Ability to handle >1 packet at a time

Non-functional Requirements:

- Open Standards and Tools
- Comprehensive configurability
- Modularity
- Low Memory Footprint ($<$ 10 KiB RAM, $<$ 30 KiB code-size)
- Low-Power Design

Outline

1. Introduction

2. **RIOT**

3. GNRC

4. Evaluation of GNRC

5. Conclusion

RIOT primer

- real-time OS for IoT (micro-kernel)
- published under LGPL at https://github.com/RIOT-OS/RIOT

RIOT primer

- real-time OS for IoT (micro-kernel)
- published under LGPL at `https://github.com/RIOT-OS/RIOT`

Scheduler:

- Tick-less scheduling policy ($O(1)$):
  - Highest priority thread runs until finished or blocked
  - ISR can preempt any thread at all time
  - If all threads are blocked or finished:
    - Special IDLE thread is run
    - Goes into low-power mode

RIOT primer

- real-time OS for IoT (micro-kernel)
- published under LGPL at https://github.com/RIOT-OS/RIOT

Scheduler:

- Tick-less scheduling policy ($O(1)$):
    - Highest priority thread runs until finished or blocked
    - ISR can preempt any thread at all time
    - If all threads are blocked or finished:
        - Special IDLE thread is run
        - Goes into low-power mode

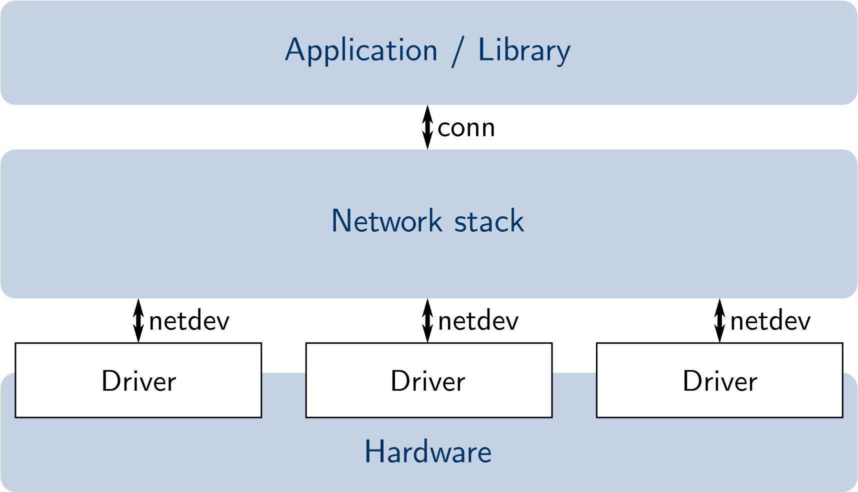
IPC:

- Synchronous (default) and asynchronous (optional, by IPC queue initialization)

Freie Universität Berlin

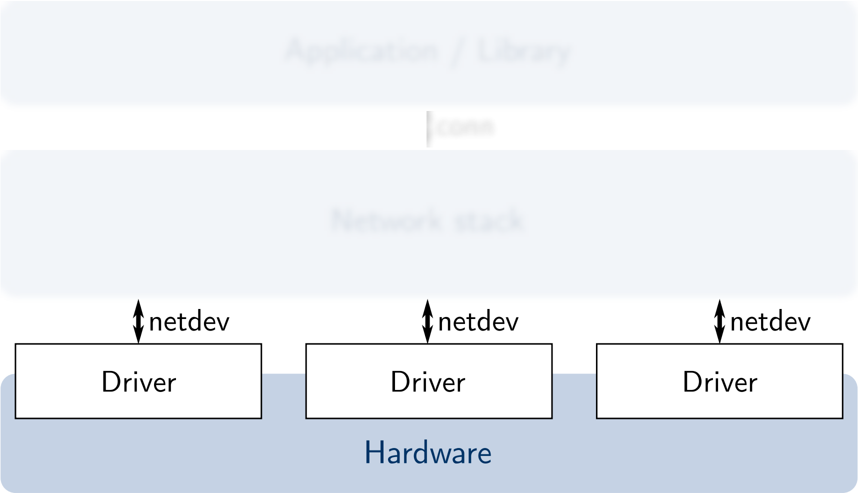
RIOT's Networking architecture

- devised to integrate any network stack into RIOT



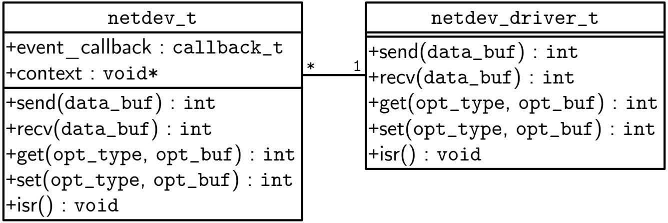
RIOT's Networking architecture

- devised to integrate any network stack into RIOT



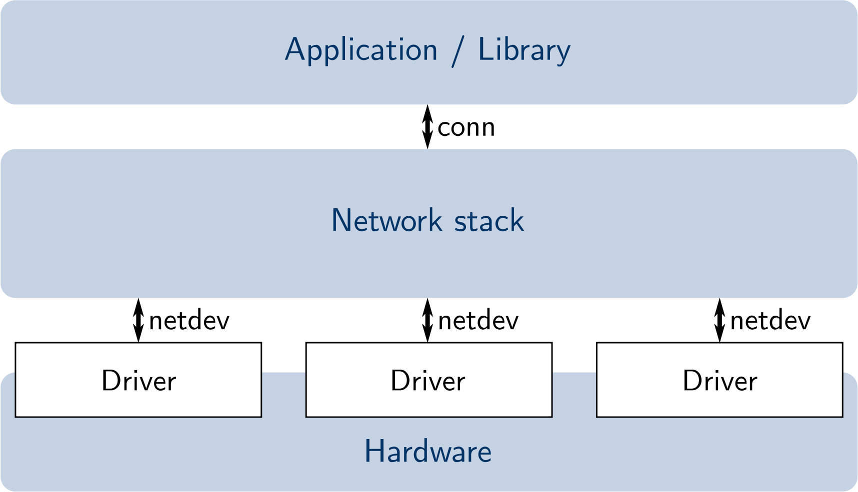
netdev

- Common network device API:

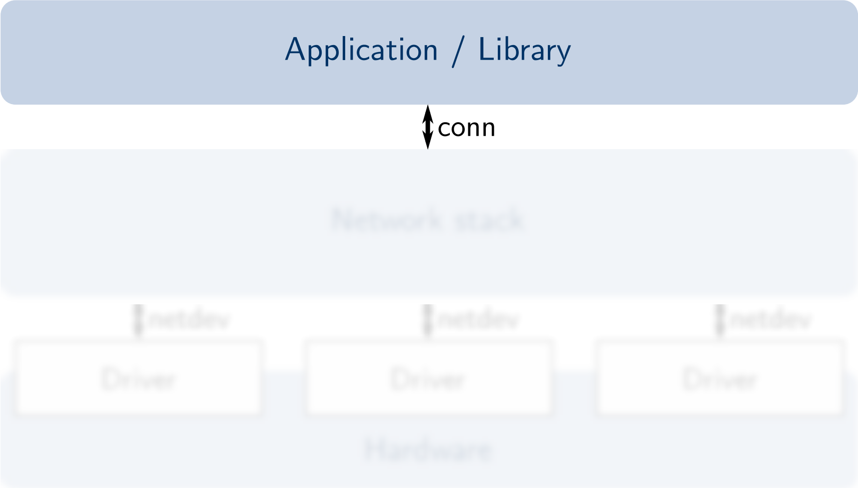


- `isr()` method allows for getting out of ISR context

RIOT's Networking architecture



RIOT's Networking architecture

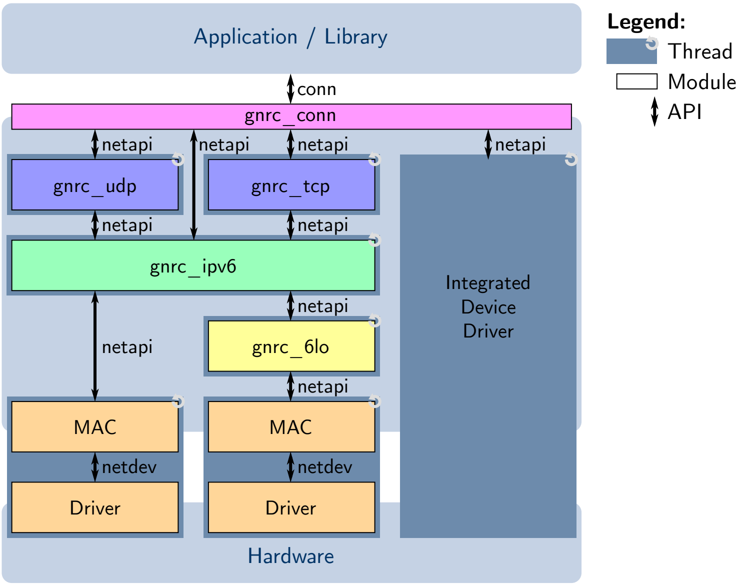


conn

- collection of unified connectivity APIs to the transport layer
- What's the problem with POSIX sockets?
  - too generic for most use-cases
  - numerical file descriptors (internal storage of state required)
  - in general: too complex for usage, too complex for porting
- protocol-specific APIs:
  - `conn_ip` (raw IP)
  - `conn_udp` (UDP)
  - `conn_tcp` (TCP)
  - ...
- both IPv4 and IPv6 supported

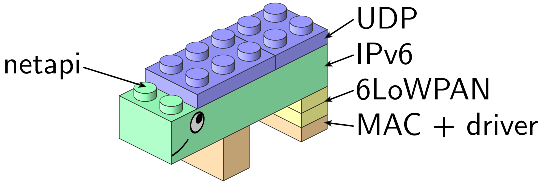
Outline

The components of GNRC

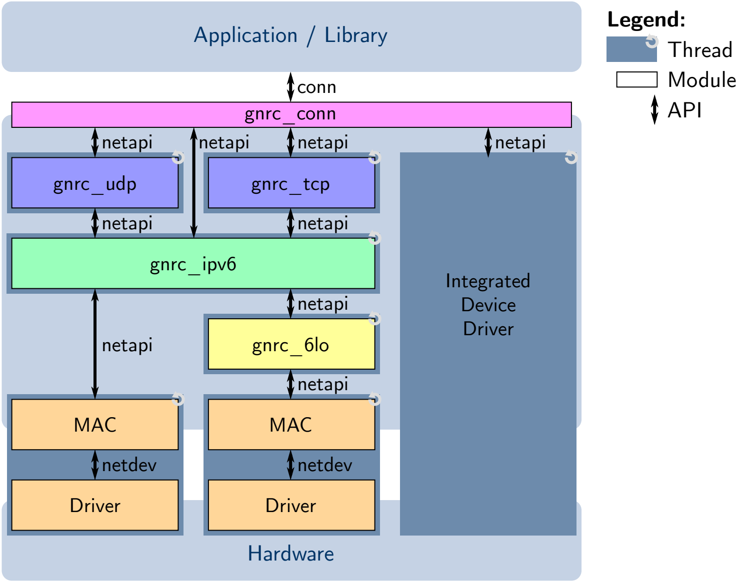


netapi

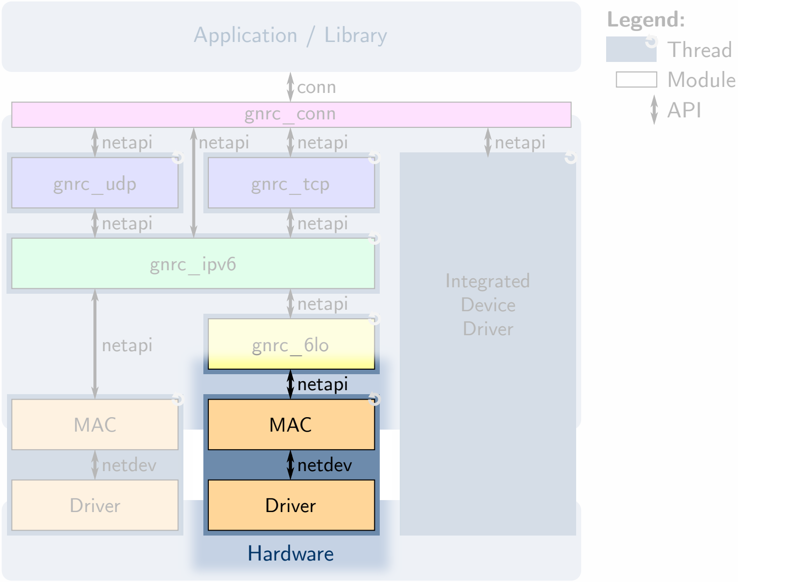
- Inter-modular API utilizing IPC
- Two asynchronous message types (don't expect reply) for data transfer:
  - GNRC_NETAPI_MSG_TYPE_SND: pass "down" the stack (send)
  - GNRC_NETAPI_MSG_TYPE_RCV: pass "up" the stack (receive)
- Two synchronous message types (expect reply) for option handling:
  - GNRC_NETAPI_MSG_TYPE_GET: get option value
  - GNRC_NETAPI_MSG_TYPE_SET: set option value
- specification deliberately vague
  $\Rightarrow$ implementations can make own preconditions on data



Network interfaces in GNRC (1)

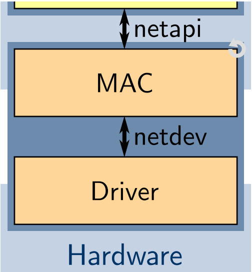


Network interfaces in GNRC (1)



Network interfaces in GNRC (2)

- `netapi`-capable thread as any other protocol implementation
- implement MAC protocol
- communication to driver via `netdev`
  ⇐ timing requirements for e.g. TDMA-based MAC protocols



netreg

- How to know where to send `netapi` messages?

netreg

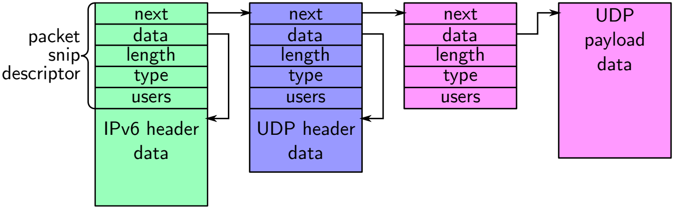
- How to know where to send `netapi` messages?
- Both protocol implementation and users can register to be interested in type + certain context (e.g. port in UDP)
  - `gnrc_netreg_register(GNRC_NETTYPE_IPV6, ALL, &me)`
  - `gnrc_netreg_register(GNRC_NETTYPE_UDP, PORT_DNS, &me)`

⇒ Find handler for packets in registry

pktbuf

- Data packet stored in `pktbuf`
- Representation: list of variable-length “packet snips”
- Protocols can *mark* sections of data to create new snip
- keeping track of referencing threads: reference counter `users`
  - if `users == 0`: packet removed from packet buffer
  - if `users > 1` and write access requested: packet duplicated (*copy-on-write*)
- to keep duplication minimal: only up to current snip

  ⇒ Reverse order of *snips* (**not data**) on reception



Outline

Feature-based comparison

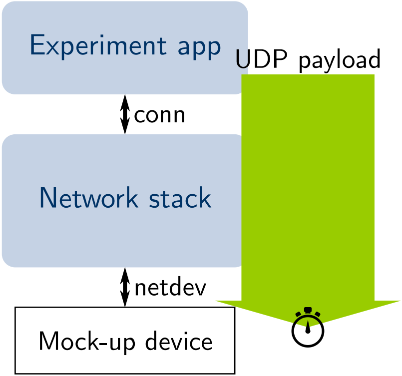
- Comparison of GNRC with emb6 (OS-independent fork of uIP) and lwIP

		6LoWPAN						ICMPv6										
		Frag.												RPL				
Stack	multi-iface.	reseq.	mult.	HC1	IPHC	NHC	IPv6	error	echo	NDP	SLAAC	6Lo-ND	MLD	st.	non-st.	TCP	UDP	CoAP
GNRC	✔	✔	✔	✘	✔	✔	✔	●	✔	✔	✘	✔	✘	✔	●	✘	✔	♦
lwIP	✔	✘	✔	✘	✔	✔	✔	✔	✔	✔	✔	✘	✔	✘	✘	✔	✔	♦
emb6	✘	✔	✘	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘	✔	✘	●	✔	✔

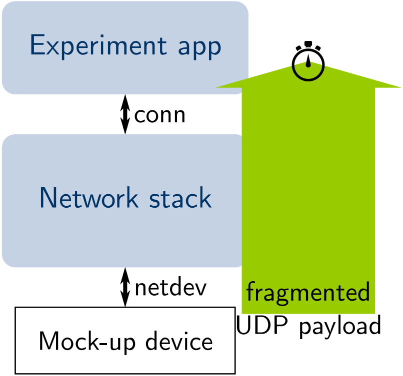
Comparison of network stack features (✔ = supported, ✘ = not supported, ● = partially supported, ♦ = support through external library)

- lwIP additionally has IPv4 (+ ARP), PPP and DNS support

Set-up for stack traversal-time tests

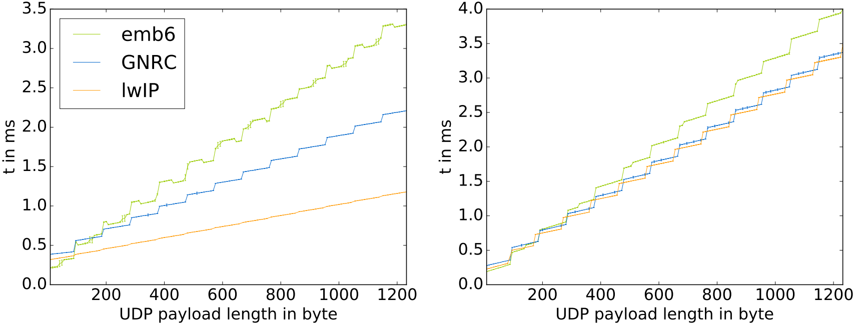


(a) UDP transmission



(b) UDP reception

Stack-traversal time

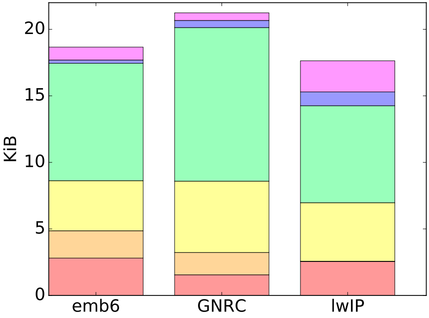


(a) UDP transmission

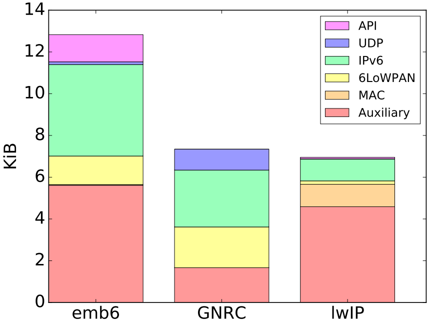
(b) UDP reception

Memory usage

- Taken application for stack traversal time tests in reception as reference
- compiled on 32-bit platform (ARM Cortex-M3)
- network stacks were configured to handle 1280 byte IPv6 packets



(a) ROM size of the stacks



(b) RAM size of the stacks

Comparison – Summary

- overall close second behind lwIP
- considering GNRC's age ($\sim$1 yr vs. $\sim$15 yr of lwIP and uIP) $\Rightarrow$ very good
- GNRC easier to work with
  - configuration of both emb6 and lwIP fiddly
  - documentation: mixed reactions from community

Discussion of GNRC

Advantages	Disadvantages
- Well defined interface enforces clear communication between modules - Use-cases are easy to describe in terms of API usage - IPC-based API allows parallel data handling per design - Very loose coupling between modules - packet buffer's size easy to adapt to given use-case	- IPC-based API is hard to debug - memory hungry due to required memory stack allocation - theory vs. praxis: cross-layer requirements everywhere

Outline

Contributions

- Co-design of GNRC, netdev, and conn
- Implementation work:
  - over 500 PRs contributed on GitHub:
    - 6LoWPAN and IPv6 (incl. NDP) layer for GNRC
    - `pktbuf`
    - several netdev-based drivers
    - port of lwIP and emb6 to RIOT
    - ...
- RIOT maintenance:
  - over 500 PRs (co-)reviewed on GitHub
  - consultance to community regarding all things GNRC
- Research:
  - co-authorship and presentation of paper to workshop @ ACM MobiSys'15
  - co-authorship of proposed paper to USENIX OSDI'16

Conclusion

- Performance-wise GNRC only (close) second after more mature lwIP
- **BUT:** GNRC developed with real-time in mind, lwIP not
- Both GNRC and emb6 can be stripped down via configuration to be smaller
- GNRC remains best candidate for **embedded RTOS** RIOT

Outlook

- Optimization efforts both size- and performance-wise
- Mitigation efforts of GNRC's disadvantages
- Expansion of GNRC's feature set.
- Further experimentation with other testing parameters
  - power consumption
  - performance under stress
  - ...
- Further experimentation with more stacks
  - BLIP (TinyOS)
  - vanilla uIP and RIME (Contiki)
  - OpenWSN
  - CCN-lite
  - ...