RIOT-Lab

How to use RIOT in the IoT-Lab

Oliver "Oleg" Hahm

INRIA

October 15, 2015

Agenda

1. Prepare for a RIOT
   - Prepare your Toolchain
   - Obtain the Code
   - Understanding RIOT
2. Using RIOT on native and the Testbed
   - Working with an Example
   - Using an IPv6 Application
3. Writing an Application for RIOT
   - Setting up the Application
   - Some helpful Features
   - Get your Hands dirty
4. Join the RIOT

Slides are online available at `http://riot-os.org/files/2015-riotlab-tutorial.pdf`.

Agenda

1. Prepare for a RIOT
   - Prepare your Toolchain
   - Obtain the Code
   - Understanding RIOT
2. Using RIOT on native and the Testbed
   - Working with an Example
   - Using an IPv6 Application
3. Writing an Application for RIOT
   - Setting up the Application
   - Some helpful Features
   - Get your Hands dirty
4. Join the RIOT

Recommended Build Environment

- For the IoT-Lab nodes we recommend to use *gcc-arm-embedded toolchain*.
  It can be found on https://launchpad.net/gcc-arm-embedded.
- A quick guide to install the proper toolchain can be found in the RIOT wiki:
  http://wiki.riot-os.org/Setup-a-Build-Environment
- See also http://wiki.riot-os.org/Board:-IoT-LAB-M3 for particular information on RIOT on the IoT-Lab nodes.
- For the *native* port you have to install 32bit libraries.
  See http://wiki.riot-os.org/Family:-native#toolchains

Github

```
git clone https://github.com/RIOT-OS/RIOT.git && \
cd RIOT && git checkout 2015.09
```

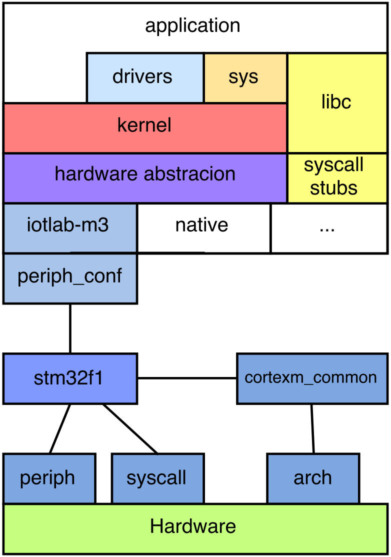
or download the zipped version:

https://github.com/RIOT-OS/RIOT/archive/2015.09.zip

The braver among you may also try the development version...

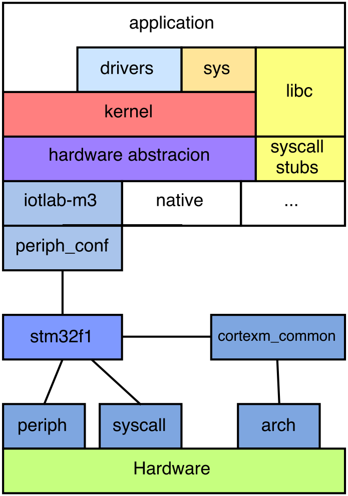
As a small fix for CLI-tools and getting rid of some warnings please go to `~/iot-lab/parts/cli-tools` and type `sudo pip install -e .[secure]`.

The Architecture



The Folder Structure

```
RIOT
 |__ boards
 |    |__ iotlab-m3
 |__ core
 |__ cpu
 |    |__ cortexm_common
 |    |__ stm32f1
 |__ dist
 |__ doc
 |__ drivers
 |    |__ at86rf231
 |    |__ isl29020
 |__ examples
 |    |__ hello-world
 |__ pkg
 |    |__ libcoap
 |__ sys
 |    |__ net
 |          |__ gnrc
 |__ tests
```



application

drivers

sys

libc

kernel

hardware abstracion

syscall stubs

iotlab-m3

native

...

periph_conf

stm32f1

cortexm_common

periph

syscall

arch

Hardware

Best Practice for RIOT programming

- Dos
  - Use static memory
  - Select the priorities carefully
  - Minimize stack usage with `DEVELHELP` and `CREATE_STACKTEST`
  - Use threads
- Donts
  - Don't use threads
  - Don't use the POSIX wrapper if implementing something from scratch
- Consult the Wiki: http://wiki.riot-os.org
- ...and the API documentation: http://doc.riot-os.org

Best Practice for RIOT programming

- Dos
  - Use static memory
  - Select the priorities carefully
  - Minimize stack usage with `DEVELHELP` and `CREATE_STACKTEST`
  - Use threads
    Increase flexibility, modularity, and robustness by using IPC.
- Donts
  - Don't use threads
    Try not to use more than one thread per module.
    Don't create threads for one-time tasks.
  - Don't use the POSIX wrapper if implementing something from scratch
- Consult the Wiki: `http://wiki.riot-os.org`
- ...and the API documentation: `http://doc.riot-os.org`

Agenda

1. Prepare for a RIOT
   - Prepare your Toolchain
   - Obtain the Code
   - Understanding RIOT
2. Using RIOT on native and the Testbed
   - Working with an Example
   - Using an IPv6 Application
3. Writing an Application for RIOT
   - Setting up the Application
   - Some helpful Features
   - Get your Hands dirty
4. Join the RIOT

Compile and run on *native*

Let's prepare a simple, virtual network:

```
[RIOT]# dist/tools/tapsetup/tapsetup -c 2
creating tapbr0
creating tap0
creating tap1
```

Now we build and start the first virtual node.

```
[RIOT]# cd examples/default/
[default]# make all term
Building application default for native w/ MCU native.
"make" -C /home/oleg/git/RIOT/cpu/native
...
RIOT native board initialized.
RIOT native hardware initialization complete.

main(): This is RIOT! (Version: 2015.09-tbilisi-HEAD)
Native RTC initialized.
Welcome to RIOT!
> help
Command              Description
---------------------------------------
reboot               Reboot the node
ps                   Prints information about running threads.
rtc                  control RTC peripheral interface
ifconfig             Configure network interfaces
txtsnd               send raw data
```

Setup the second Node

Ok, how about a second node?
Therefore, we specify the *tap* interface to use by (re-)using the PORT environment variable. We can use any created *tap* interface from the previous step. (Default is *tap0*.) Once the node has started, we check its radio address.

```
[default]# PORT=tap1 make term
Welcome to RIOT!
> ifconfig
Iface  4   HWaddr: 8a:c2:b6:72:eb:57
           Source address length: 6
```

Connecting the two Nodes

It's time to send our first packet (assuming that the first node is still running.). Go back to the first node and type:

```
> txtsnd 4 8a:c2:b6:72:eb:57 riotlab
```

On the second node we should see something like:

```
PKTDUMP: data received:
~~ SNIP  0 - size:   7 byte, type: NETTYPE_UNDEF (0)
000000 72 69 6f 74 6c 61 62
~~ SNIP  1 - size:  20 byte, type: NETTYPE_NETIF (-1)
if_pid: 4  rssi: 0  lqi: 0
src_l2addr: a6:b7:d0:ea:de:f9
dst_l2addr: 8a:c2:b6:72:eb:57
~~ PKT    -  2 snips, total size:  27 byte
```

Using the same example on the testbed

Running the same example on a real node is very similar.
We use the environment variable BOARD to specify the target platform – and IOTLAB_ variables for testbed configuration.

```
[default]# BOARD=iotlab-m3 IOTLAB_SITE=lille IOTLAB_DURATION=60 make all
    iotlab-exp
Building application "default" for "iotlab-m3" with MCU "stm32f1"
"make" -C /tmp/RIOT/boards/iotlab-m3
...
Waiting that experiment 29409 gets in state Running
"Running"
```

And now connect to the nodes using the serial_aggregator:

```
[default]# BOARD=iotlab-m3 IOTLAB_SITE=lille make iotlab-term
Connection to lille.iot-lab.info closed.
1444752645.849845;Aggregator started
ifconfig
1444752749.523268;m3-13;ifconfig
1444752749.523673;m3-9;ifconfig
1444752749.525317;m3-9;Iface  4   HWaddr: 9d:12  Channel: 26  NID: 0x23
    TX-Power: 0dBm  State: IDLE CSMA Retries: 4
1444752749.525976;m3-13;Iface  4   HWaddr: 96:16  Channel: 26  NID: 0x23
     TX-Power: 0dBm  State: IDLE CSMA Retries: 4
...
```

The Shell in a Nutshell

- For this step we will use *gnrc_networking* application from examples directory.
- You can configure RIOT to provide you with some default system shell commands.
- All available shell commands and some online help are shown by calling `help`:

```
> help
help
Command              Description
---------------------------------------
udp                  send data over UDP and listen on UDP ports
reboot               Reboot the node
ps                   Prints information about running threads.
ping6                Ping via ICMPv6
mersenne_init        initializes the PRNG
mersenne_get         returns 32 bit of pseudo randomness
ifconfig             Configure network interfaces
txtsnd               send raw data
fibroute             Manipulate the FIB (info: 'fibroute [add|del]')
ncache               manage neighbor cache by hand
routers              IPv6 default router list
rpl                  rpl configuration tool [help|init|rm|root|show]
```

The selection of commands depends on the configuration of your application (and may vary a little bit for different platforms).

Let's communicate

The *gnrc_networking* application provides you with some helpful shell commands:

1. ping: The famous ICMP diagnosis tool (system command).
2. udp: A very basic command for arbitrary UDP connections (application command).

After we flashed two nodes, we'll have to find out their IPv6 addresses using `ifconfig`:

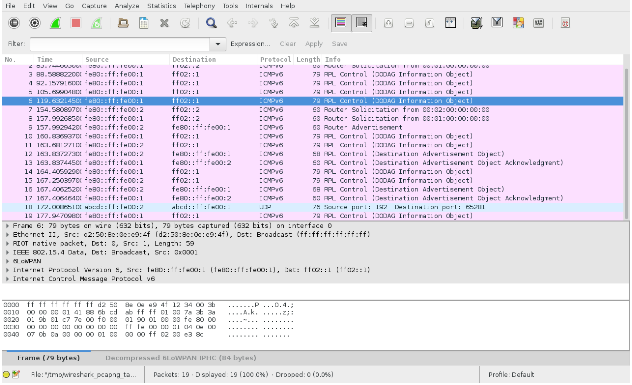
```
    m3-66;ifconfig
Iface  7   HWaddr: 7f:06  Channel: 26  NID: 0x23  TX-Power: 0dBm  State: IDLE
     CSMA Retries: 4
   Long HWaddr: 36:32:48:33:46:d8:7f:06
   AUTOACK  CSMA  MTU:1280  6LO  IPHC
   Source address length: 8
   Link type: wireless
   inet6 addr: ff02::1/128  scope: local [multicast]
   inet6 addr: fe80::3432:4833:46d8:7f06/64  scope: local
   inet6 addr: ff02::1:ffd8:7f06/128  scope: local [multicast]
```

Now we can try to ping the other node:

```
    > ping fe80::ff:fe00:f01e
    INFO # ping fe80::ff:fe00:f01e
    INFO # Echo reply from fe80::ff:fe00:f01e received, rtt: 0.0340s
```

Virtualize your Network

- You can also monitor the traffic with *Wireshark*.
- There's a python framework to setup arbitrary topologies using defined loss-rates: http://wiki.riot-os.org/Virtual-riot-network



```
No.  Time        Source            Destination       Protocol Length Info
   2 83.7446030( fe80::ff:fe00:1   ff02::2           ICMPv6   60 Router Solicitation from 00:01:00:00:00:00
   3 88.5888220( fe80::ff:fe00:1   ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
   4 92.1579160( fe80::ff:fe00:1   ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
   5 105.6990480( fe80::ff:fe00:1  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
   6 119.6321450( fe80::ff:fe00:1  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
   7 154.5808970( fe80::ff:fe00:2  ff02::2           ICMPv6   60 Router Solicitation from 00:02:00:00:00:00
   8 157.9926850( fe80::ff:fe00:1  ff02::2           ICMPv6   60 Router Solicitation from 00:01:00:00:00:00
   9 157.9929420( fe80::ff:fe00:2  fe80::ff:fe00:1   ICMPv6   60 Router Advertisement
  10 160.8369370( fe80::ff:fe00:1  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
  11 163.6812710( fe80::ff:fe00:2  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
  12 163.8372730( fe80::ff:fe00:2  fe80::ff:fe00:1   ICMPv6   68 RPL Control (Destination Advertisement Object)
  13 163.8374450( fe80::ff:fe00:1  fe80::ff:fe00:2   ICMPv6   60 RPL Control (Destination Advertisement Object Acknowledgment)
  14 164.4059290( fe80::ff:fe00:1  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
  15 167.2503970( fe80::ff:fe00:2  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
  16 167.4062520( fe80::ff:fe00:2  fe80::ff:fe00:1   ICMPv6   68 RPL Control (Destination Advertisement Object)
  17 167.4064640( fe80::ff:fe00:1  fe80::ff:fe00:2   ICMPv6   60 RPL Control (Destination Advertisement Object Acknowledgment)
  18 172.0086510( abcd::ff:fe00:2  abcd::ff:fe00:1   UDP      76 Source port: 192  Destination port: 65281
  19 177.9470980( fe80::ff:fe00:1  ff02::1           ICMPv6   79 RPL Control (DODAG Information Object)
```

```
▸ Frame 6: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0
▸ Ethernet II, Src: d2:50:8e:0e:e9:4f (d2:50:8e:0e:e9:4f), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ RIOT native packet, Dst: 0, Src: 1, Length: 59
▸ IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0001
▸ 6LoWPAN
▸ Internet Protocol Version 6, Src: fe80::ff:fe00:1 (fe80::ff:fe00:1), Dst: ff02::1 (ff02::1)
▸ Internet Control Message Protocol v6
```

```
0000  ff ff ff ff ff ff d2 50  8e 0e e9 4f 12 34 00 3b   .......P ...O.4.;
0010  00 00 00 01 41 88 6b cd  ab ff ff 01 00 7a 3b 3a   ....A.k. .....z;:
0020  01 9b 01 c7 7e 00 f0 00  01 90 01 00 00 fe 80 00   ....~... ........
0030  00 00 00 00 00 00 00 00  ff fe 00 00 01 04 0e 00   ........ ........
0040  07 0b 0a 00 00 00 01 00  00 00 ff 02 00 e3 8c      ........ .......
```

File: "/tmp/wireshark_pcapng_ta... | Packets: 19 · Displayed: 19 (100.0%) · Dropped: 0 (0.0%)

Agenda

1. Prepare for a RIOT
   - Prepare your Toolchain
   - Obtain the Code
   - Understanding RIOT
2. Using RIOT on native and the Testbed
   - Working with an Example
   - Using an IPv6 Application
3. Writing an Application for RIOT
   - Setting up the Application
   - Some helpful Features
   - Get your Hands dirty
4. Join the RIOT

The Makefile

To create your own RIOT example, you need only two files:

1. A C (or C++) file with a main function
2. A Makefile

You can find a template for a Makefile in `dist/Makefile`.

```
# Giving your application a name
APPLICATION = riotlab1
# Choosing a default platform
BOARD ?= native
# Specifying the RIOT folder
RIOTBASE ?= $(CURDIR)/../../RIOT
# Some helpful compiler flags
CFLAGS += -DSCHEDSTATISTICS -DDEVELHELP
# Quieten the building process
QUIET ?= 1
# Modules to include:
USEMODULE += posix
USEMODULE += xtimer
USEMODULE += shell_commands
# Let RIOT's build system take care of the rest
include $(RIOTBASE)/Makefile.include
```

The `main()` Function

The only mandatory thing in your application is a `main` function with this prototype:

```
int main(void)
```

Standard C libraries can be used and parts of POSIX are ble through a wrapper within RIOT.

```
#include <stdio.h>
#include <unistd.h>

#define WAIT_USEC   (1000 * 1000)

int main(void)
{
    puts("Hello!");
    usleep(WAIT_USEC);
    puts("Good night!");

    return 0;
}
```

Doing it the RIOT way

Instead of POSIX functions is usually advisable (and more efficient) to just use the native RIOT functions:

```
#include <stdio.h>
#include "xtimer.h"


int main(void)
{
    puts("Let's throw a brick!");
    xtimer_usleep(SEC_IN_USEC);
    puts("System terminated");

    return 0;
}
```

Starting the Shell

```
#include "shell_commands.h"

static int my_echo(int argc, char **argv);
...
const shell_command_t shell_commands[] = {
    {"echo", "Echo the user’s input", my_echo},
    {NULL, NULL, NULL}
};
...
/* allocate some memory for the input line */
char line_buf[SHELL_DEFAULT_BUFSIZE];
/* starting the shell loop (blocking) */
shell_run(shell_commands, line_buf, SHELL_DEFAULT_BUFSIZE);
```

Threads and IPC

```
/* allocate memory for the thread's stack */
char my_stack[THREAD_STACKSIZE_MAIN];
/* define a function as entry point for your new thread */
void *my_thread(void *arg) {
...
kernel_pid_t pid = thread_create(my__stack, sizeof(my_thread_stack),
                        THREAD_PRIORITY_MAIN - 1, CREATE_STACKTEST,
                        my_thread, NULL, "mythread");

msg_t m;
m.content.value = 1;
msg_send_receive(&m, &m, pid);
```

Now it's up to you

Your task is now to extend the *sixlowapp* example.

- Check the source code of the `posix_sockets` example and API documentation online: `http://doc.riot-os.org`
- Check the documentation for the sensor API, e.g. for the light sensor: `http://doc.riot-os.org/group__driver__isl29020.html`
- Initialize all four sensors in the beginning of the application.
- Extend the netcat command to send optionally measured sensor data.
- Print the measurements on the receiving side.

Agenda

1. Prepare for a RIOT
   - Prepare your Toolchain
   - Obtain the Code
   - Understanding RIOT
2. Using RIOT on native and the Testbed
   - Working with an Example
   - Using an IPv6 Application
3. Writing an Application for RIOT
   - Setting up the Application
   - Some helpful Features
   - Get your Hands dirty
4. Join the RIOT

Where to get help

- Mailing Lists
  - devel@riot-os.org
  - users@riot-os.org
- IRC on irc.freenode.org, #riot-os
- Regular video conferencing meetings

Thank you!
Any questions?