Broadcasting in Contiki-OS

Shantanoo Desai
prepared for:
Prof. Dr. Anna Förster

Sustainable Communication Networks
University of Bremen

November 20, 2015

Outline

Contents of this section

Rime Stack

- “Just send data don’t worry whether corrupted or lost!"
- Broadcast: Send the same message to everyone

Contiki-OS uses **Rime-stack** which is:

- Lightweight, layered communication stack for Sensor networks

Why **RIME**? – traditional communication stack TOO Stringent to apply on sensor node

Rime stack in Contiki

- Main source files for Rime stack found in : *contiki-2.7/core/net/rime*
- Predefined examples found in : *contiki-2.7/examples/rime*

Inside the */rime* folder, available examples :

- broadcast (both: .c and .csc(Cooja Simulator))
- unicast
- collect
- mesh
- multihop

[**HINT**: use the ‘ls’ command in the folder]

UDP Broadcast Example

Assuming you in the *contiki-2.7/example/rime* folder do the following:

1. open the **example-broadcast.c** file using gedit

   ```
   /examples/rime/ gedit example-broadcast.c
   ```

2. find *packetbuf_copyfrom()* function in **PROCESS_THREAD** section
3. replace “Hello” to your name and change the subsequent number to the Length of your name + 1(for the null character)
   e.g. *packetbuf_copyfrom(“John", 5)*
4. Save the file and compile

   ```
   make TARGET=sky savetarget
   ```

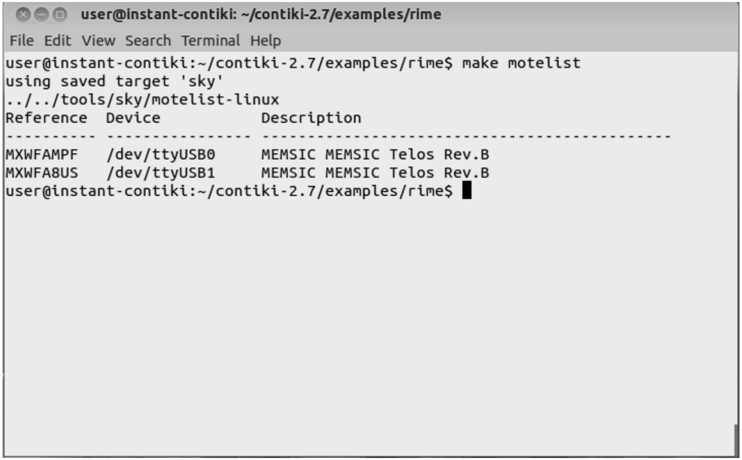
Broadcast on Tmote Sky

NOTE: When connecting two or more motes to the Virtual Machine:

1. Check the connection of motes by clicking on Virtual Machine tab on VMware player & check Removable devices section and make sure all the motes are connected by clicking on Connect(disconnect from host)
2. In terminal to show which motes are connected on which USB ports:

```
make motelist
```

Checking Multiple Connected Sky Motes


user@instant-contiki: ~/contiki-2.7/examples/rime
File Edit View Search Terminal Help

```
user@instant-contiki:~/contiki-2.7/examples/rime$ make motelist
using saved target 'sky'
../../tools/sky/motelist-linux
Reference  Device           Description
---------- ---------------- ---------------------------------------------
MXWFAMPF   /dev/ttyUSB0     MEMSIC MEMSIC Telos Rev.B
MXWFA8US   /dev/ttyUSB1     MEMSIC MEMSIC Telos Rev.B
user@instant-contiki:~/contiki-2.7/examples/rime$ 
```

UDP Broadcast on Tmote Sky

For programming individual motes separately use:

```
make TARGET=sky savetarget
make example-broadcast.upload MOTE=1
```

MOTE=1 will program the Sky mote at */dev/ttyUSB0* without programming the Sky mote at *dev/ttyUSB1*
– Try for the mote connected at *dev/ttyUSB1* [HINT: MOTE=2]
To Observe Output:

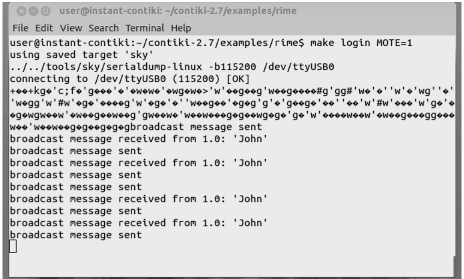
```
make login MOTE=1
make login MOTE=2 (in a separate NEW Terminal)
```

Contents of this section

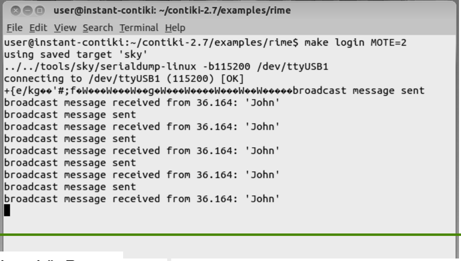
Broadcast Results

Broadcast Output

Observe the Broadcast Message “John" sent by both the motes and received to each other with different addresses.


user@instant-contiki: ~/contiki-2.7/examples/rime
File Edit View Search Terminal Help

```
user@instant-contiki:~/contiki-2.7/examples/rime$ make login MOTE=1
using saved target 'sky'
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
+◆◆+kg◆'c;f◆'g◆◆◆'◆'◆w◆w◆'◆wg◆w◆>'w'◆◆g◆◆g'w◆◆g◆◆◆◆#g'gg#'w◆'◆''w'◆'wg''◆'
'w◆gg'w'#w'◆g◆'◆◆◆◆g'w'◆g◆'◆''w◆◆g◆◆'◆g◆g'g'◆'g◆◆g◆'◆◆''◆◆'w'#w'◆◆◆'w'g◆'◆
◆g◆wgw◆◆w'◆w◆◆g◆◆w◆◆g'gw◆◆w◆'w◆◆w◆◆◆g◆g◆◆wg◆g◆'g◆'w'◆◆◆◆w◆◆w'◆w◆◆g◆◆◆gg◆◆◆
w◆◆'w◆◆w◆◆g◆g◆◆g◆g◆gbroadcast message sent
broadcast message received from 1.0: 'John'
broadcast message sent
broadcast message received from 1.0: 'John'
broadcast message sent
broadcast message sent
broadcast message received from 1.0: 'John'
broadcast message sent
broadcast message received from 1.0: 'John'
broadcast message sent
```


user@instant-contiki: ~/contiki-2.7/examples/rime
File Edit View Search Terminal Help

```
user@instant-contiki:~/contiki-2.7/examples/rime$ make login MOTE=2
using saved target 'sky'
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB1
connecting to /dev/ttyUSB1 (115200) [OK]
+{e/kg◆◆'#;f◆W◆◆◆W◆◆◆W◆◆g◆W◆◆◆W◆◆◆◆W◆◆◆W◆◆W◆◆◆◆◆broadcast message sent
broadcast message received from 36.164: 'John'
broadcast message sent
broadcast message received from 36.164: 'John'
broadcast message sent
broadcast message received from 36.164: 'John'
broadcast message sent
broadcast message received from 36.164: 'John'
broadcast message sent
broadcast message received from 36.164: 'John'
```

Contents of this section

Code Understanding

Understanding the Code

Headers:

```
#include "contiki.h" /* For contiki apps */
#include "net/rime.h" /* For RIME stack */
#include <stdio.h> /* For printf()*/
```

Process Macro: for making application specific macros in contiki

1. name a PROCESS
2. AUTOSTART the Process

```
PROCESS(broadcast_process,"Broadcast example");
AUTOSTART_PROCESSES(&example_broadcast_process);
```

Understanding the code

For complete operations of functions refer to *core/net/rime/broadcast.c* and *core/net/rime/broadcast.h*
Observe the *broadcast_recv()* function:

```
broadcast_recv(struct broadcast_conn*,
const rimeaddr_t*);
```

Function: parses an incoming packet and displays the message and the address of the sender.

- *struct broadcast_conn* *: This structure which has 2 structures : abc_conn, broadcast_callbacks *. The abc_conn is basic type of connection over which the broadcast connection is developed. And, the broadcast_callbacks point to recv and sent functions (in this example, just recv)
- rimeaddr_t *: This is a union which has a character array u8[RIMEADDR_SIZE].

Understanding the code

broadcast connections

```
broadcast_close(struct broadcast_conn *)
broadcast_open(struct broadcast_conn *, uint16_t ,
const struct broadcast_callbacks *)
```

- broadcast_close(struct broadcast_conn *): for closing a previously open *best-effort* connection for broadcasting messages
- broadcast_open(struct broadcast_conn *, uint16_t ,const struct broadcast_callbacks *): to open a *best effort* broadcasting UDP port
  - broadcast_conn : A pointer to a struct broadcast_conn
  - uint16_t: The channel on which the connection will operate
  - broadcast_callbacks : A struct broadcast_callbacks with function pointers to functions that will be called when a packet has been received

TIMERS in contiki Applications

```
etimer_set(struct etimer *, clock_time_t)
```

Function: set an event timer for a time sometime in the future. When the event timer expires, the event PROCESS_EVENT_TIMER will be posted to the process that called the *etimer_set()* function

Contents of this section

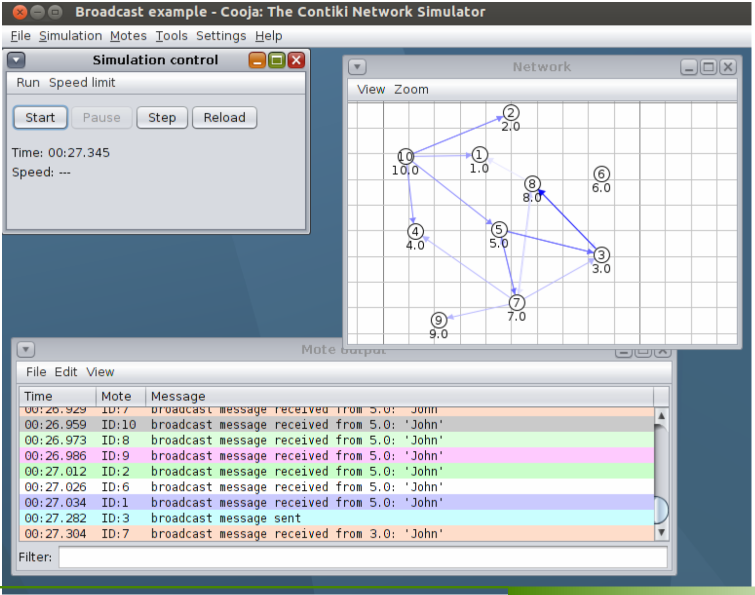
Broadcast in Cooja

Cooja Simulation of UDP Broadcast

- open the **example-broadcast.csc** in Cooja simulator
  - In Cooja Simulator go to **File** – **Open Simulation** – **Browse**
  - Navigate to *examples/rime* – select **example-broadcast.csc**

Simulation Environment has 10 motes in the Network panel
For traffic visibility click on **View** in Network Panel and check on for **Radio Traffic** and Click on Start

UDP broadcast Simulation



```
Time       Mote   Message
00:26.929  ID:7   broadcast message received from 5.0: 'John'
00:26.959  ID:10  broadcast message received from 5.0: 'John'
00:26.973  ID:8   broadcast message received from 5.0: 'John'
00:26.986  ID:9   broadcast message received from 5.0: 'John'
00:27.012  ID:2   broadcast message received from 5.0: 'John'
00:27.026  ID:6   broadcast message received from 5.0: 'John'
00:27.034  ID:1   broadcast message received from 5.0: 'John'
00:27.282  ID:3   broadcast message sent
00:27.304  ID:7   broadcast message received from 3.0: 'John'
```

References

for Rime Stack:
`http://dunkels.com/adam/dunkels07rime.pdf`
For Timers in Contiki-OS: `https://github.com/contiki-os/contiki/wiki/Timers`