

Trickle & Java

Giacomo Tanganelli
PhD student @ University of Pisa
g.tanganelli@iet.unipi.it

Recall Trickle

- Each node maintains a counter c and a timer t in range $[1/2, 1]$ (at start, $l = l_{\min}$)
- When a node receives metadata that is “consistent”, it increments c
- At time t , the node broadcasts a summary of its metadata if $c < K$ (redundancy threshold)
- When the interval l expires
 - l is doubled (up to l_{\max})
 - c is reset to zero
 - t is reset to a new value in the range $[1/2, 1]$
- When a node receives metadata that is “inconsistent” (and $l > l_{\min}$), l is reset to l_{\min} and c and t are reset

Contiki Trickle

- All the RPL configuration parameters are in:
 - `core/net/rpl/rpl-conf.h`
- `K = RPL_CONF_DIO_REDUNDANCY`
- `Imin = RPL_CONF_DIO_INTERVAL_MIN`
 - $2^{RPL_CONF_DIO_INTERVAL_MIN}$
- `Imax = RPL_CONF_DIO_INTERVAL_DOUBLINGS`
 - $2^{(RPL_CONF_DIO_INTERVAL_MIN + RPL_CONF_DIO_INTERVAL_DOUBLINGS)}$

Change parameters

- In order to change an RPL parameter edit your Makefile:

```
CFLAGS+= -DRPL_CONF_DIO_REDUNDANCY=1
```




Display RPL output

- To perform some analysis you must modify source file inside `core/net/rpl/`
- Set `DEBUG DEBUG_PRINT` in `rpl-timers.c` to investigate Trickle.
- Can also set custom `printf` in order to detect custom events.

Exercise 1

1. Modify `rpl-timers.c`
 - Insert a `printf` before: `dio_output(instance, NULL);`
2. Modify the Makefile of `example/ipv6/rpl-udp`
 - `CFLAGS+= -DRPL_CONF_DIO_REDUNDANCY=1`
3. Create a simulation with an `udp-server` and 5 `udp-client` (`example/ipv6/rpl-udp/`)
4. Check if your message is displayed.
5. Save your simulation.

Simulation

- COOJA can be used also as a simulation environment.
- Let's open Tools -> Simulation Script Editor
- The Simulation Script Editor is used to manage a simulation with or without the COOJA's GUI.
- Main variables:
 - id: the mote who has sent the message
 - msg: the message sent
 - time: when the message has been sent

Simulation (2)

- Main functions:
 - TIMEOUT(sec) : set when the simulation must end.
 - YIELD() : waits for the next message.
 - log.log(message): print a message on the script console.
 - write(Mote mote, String msg): write to the mote's serial interface.
 - WAIT_UNTIL(expr): yield if expr is not true.

Example Script

```
TIMEOUT(72000);  
i=0;  
while(i<100){  
    log.log(time+": "+id+": "+msg);  
    YIELD();  
    i++;  
}
```

- Once the script is written click on:
 - Run -> Activate

Exercise 2

- Try the last exercise with a custom version of the example script which print out time:id:msg only when the message is the one you defined inside rpl-timers.h.
- NOTE: to check the message you can use:
 - `YIELD_THEN_WAIT_UNTIL(msg.equals("your message"))`

COOJA without GUI

- In order to run a lot of simulation the GUI is too slow.
- Execute *make TARGET=z1 clean* inside your application folder.
- Execute your simulation by:
 - `java -jar $CONTIKI_HOME/tools/cooja/dist/cooja.jar -nogui=$TEST.csc -contiki=$CONTIKI_HOME`

COOJA without GUI (2)

- Once the simulation finish you will have two log file:
 - COOJA.log
 - COOJA.testlog
- The second one is the output generated by your script.

Exercise 3

- Run the simulation of exercise 2 without GUI two times. The first time with a `RPL_CONF_DIO_REDUNDANCY=1`, while the second time with a `RPL_CONF_DIO_REDUNDANCY=10`.
- NOTE: Save the logs in a separate folder between executions or they will be overwritten.
- NOTE: Remember to execute make clean.

Java UDP

```
import java.net.*;
```

- Set a socket:
 - `DatagramSocket(int client_port);`
- Set address:
 - `InetAddress.getByName(String host);`
- Set a packet:
 - `DatagramPacket(byte[] buf, int length, InetAddress ip, int server_port);`
- Send a packet:
 - `DatagramSocket.send(DatagramPacket p);`
- Set a packet for receive:
 - `DatagramPacket(byte[] buf, int length);`
- Receive a packet:
 - `DatagramSocket.receive(DatagramPacket p);`

Exercise 4

- Write a java udp client which sends a message to the udp receiver running inside cooja.
 - Modify the previous example in order to have an ack from the receiver.

- NOTE:

```
int simple_udp_register (struct simple_udp_connection *c, uint16_t local_port,  
uip_ipaddr_t *remote_addr, uint16_t remote_port, simple_udp_callback  
receive_callback)
```

```
simple_udp_register(&unicast_connection,UDP_PORT, NULL, 0, receiver);
```

- NOTE: a connection is one-way.