UPPSALA
UNIVERSITET

Property-Based Testing of Sensor Networks

Andreas Löscher

andreas.loscher@it.uu.se

Uppsala University

UPPSALA UNIVERSITET

Sensor Network Testing is Important

- Integral to Software Development
- Sensor networks are pushing into the commercial domain
- Failure can affect the whole network
- Used in critical domains:
  - Health Care
  - Process Control

UPPSALA UNIVERSITET

Testing the Encoder and Decoder of a Protocol

- Functions: $encode()$ and $decode()$
- Does decoding an encoded message yield the original message?
- Test it!

UPPSALA UNIVERSITET

Property-Based Testing

- We specify:
  - Generic Structure of the Input
  - General properties for valid system behaviour
- A PBT tool automatically tests these properties
  - Generate wide range of input
  - Run the system under test with the generated input
  - Check the system against properties

UPPSALA UNIVERSITET

Example

```
prop_encode_decode() ->
  ?FORALL(I, input(),
   I == protocol:decode(protocol:encode(I))).


input() ->
  list(range(32, 127)).
```

- The input `I` is randomly **generated**
- The test code is run for each input
- The property is checked for each test instance

UPPSALA UNIVERSITET

Demo

UPPSALA UNIVERSITET

Testing Sensor Networks

- Implemented in C
  - Hardware dependent
- Distributed Systems
  - Network Topologies
  - Heterogeneous Hardware
- Functional and Non-Functional Properties
  - Energy Consumption
  - Timing

UPPSALA UNIVERSITET

Testing the Encoder and Decoder of a Protocol implemented in C

- Functions:

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

- Does decoding an encoded message yield the original message?
- Test it!

UPPSALA UNIVERSITET

Nifty

- NIF Interface Generator
- http://parapluu.github.io/nifty/

UPPSALA UNIVERSITET

Demo

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),
```

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),

      %% pointers for the encoded and decoded message
      Encoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
      Decoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
```

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),

      %% pointers for the encoded and decoded message
      Encoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
      Decoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),

      %% encode
      0 = c_protocol:encode(Message, Encoded),

      %% decode
      0 = c_protocol:decode(nifty:dereference(Encoded), Decoded),
```

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),

      %% pointers for the encoded and decoded message
      Encoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
      Decoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),

      %% encode
      0 = c_protocol:encode(Message, Encoded),

      %% decode
      0 = c_protocol:decode(nifty:dereference(Encoded), Decoded),

      %% C string -> Erlang List
      ProcessedMessage = nifty:cstr_to_list(nifty:dereference(Decoded)),
```

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),

      %% pointers for the encoded and decoded message
      Encoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
      Decoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),

      %% encode
      0 = c_protocol:encode(Message, Encoded),

      %% decode
      0 = c_protocol:decode(nifty:dereference(Encoded), Decoded),

      %% C string -> Erlang List
      ProcessedMessage = nifty:cstr_to_list(nifty:dereference(Decoded)),

      %% cleanup
      nifty:free([Encoded, Decoded, Message, …]),
```

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

```
prop_c_impl() ->
  ?FORALL(I, input(),
    begin
      %% Erlang List -> C string
      Message = nifty:list_to_cstr(I),

      %% pointers for the encoded and decoded message
      Encoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),
      Decoded = nifty:as_type(nifty:pointer(), "c_protocol.char **"),

      %% encode
      0 = c_protocol:encode(Message, Encoded),

      %% decode
      0 = c_protocol:decode(nifty:dereference(Encoded), Decoded),

      %% C string -> Erlang List
      ProcessedMessage = nifty:cstr_to_list(nifty:dereference(Decoded)),

      %% cleanup
      nifty:free([Encoded, Decoded, Message, …]),

      %% result
      I =:= ProcessedMessage
    end).
```

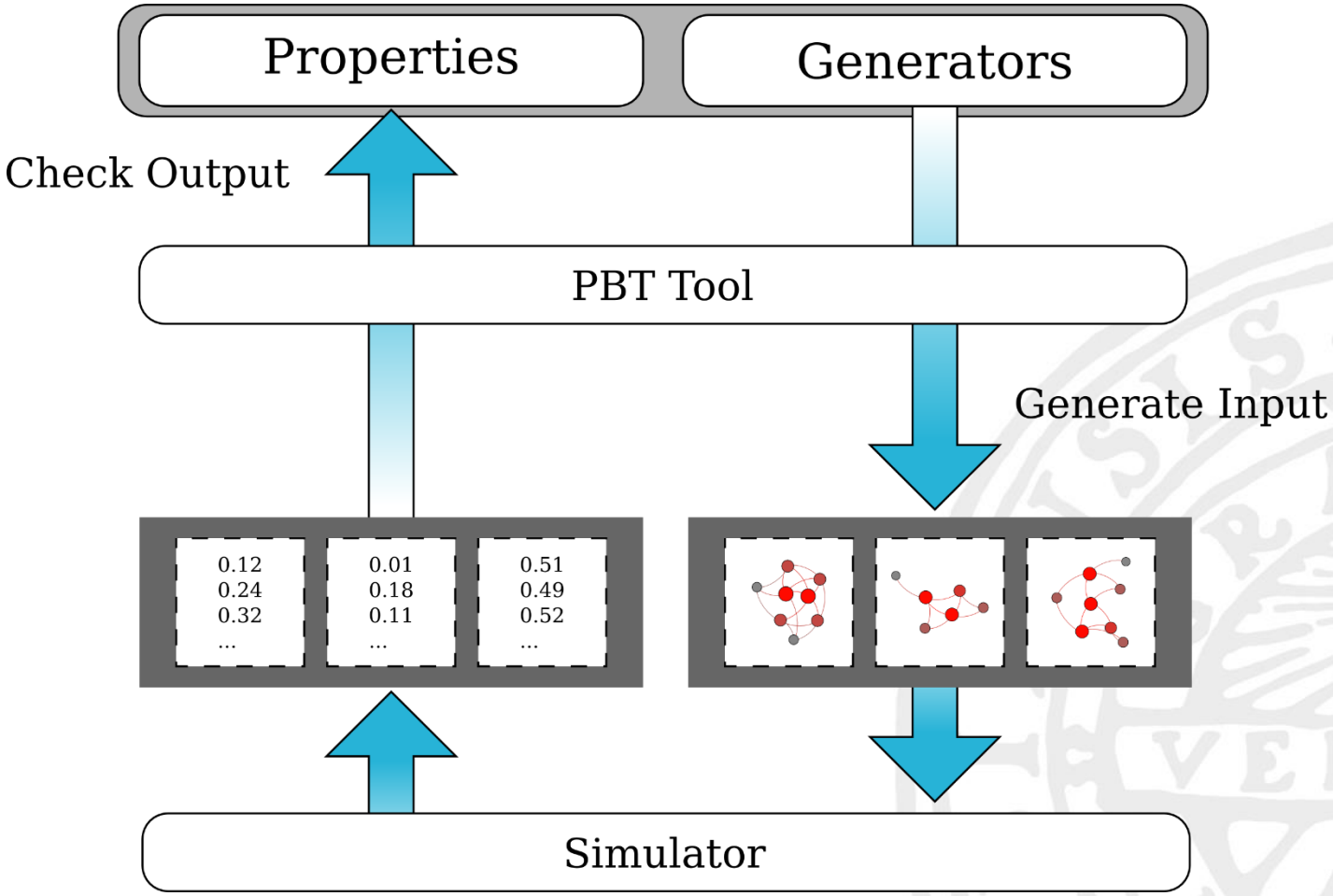
```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

UPPSALA UNIVERSITET

Demo

UPPSALA UNIVERSITET

Framework



UPPSALA UNIVERSITET

Duty-Cycle of X-MAC

- Setup:
  - Random distribution of UDP server and client nodes
  - Client nodes sends periotically messages to server nodes
  - IPv6 and RPL
- Test:
  - Has X-MAC for any network a duty-cycle > 10%?

UPPSALA UNIVERSITET

Property

```
prop_xmac() ->
  ?FORALL(Motes, motes(),
```

- Generates a random configuration of motes
- Motes:
  - Position (x,y, 0)
  - Mote with ID 2 is server
  - Other motes are clients

```
mote() ->
  tuple([float(0, 100),
         float(0, 100),
         0.0]).

motes() ->
  ?SUCHTHAT(Motes, list(mote()),
            length(Motes)>=2).
```

UPPSALA UNIVERSITET

Property

```
prop_xmac() ->
  ?FORALL(Motes, motes(),
    begin
      Handler = nifty_cooja:start("…", "…xmac.csc", []),
      Mote_IDs = add_motes(Handler, Motes),
```

- Start and initialize the simulation

```
add_mote(Handler, {Pos, Type}) ->
  {ok, ID} = nifty_cooja:mote_add(Handler, Type),
  ok = nifty_cooja:mote_set_pos(Handler, ID, Pos),
  ok = nifty_cooja:mote_hw_listen(Handler, ID),
  ID.
```

UPPSALA UNIVERSITET

Property

```
prop_xmac() ->
  ?FORALL(Motes, motes(),
    begin
      Handler = nifty_cooja:start("…", "…xmac.csc", []),
      Mote_IDs = add_motes(Handler, Motes),
      ok = nifty_cooja:simulation_step(Handler, 120000),
```

- Run the simulation

UPPSALA UNIVERSITET

Property

```
prop_xmac() ->
  ?FORALL(Motes, motes(),
    begin
      Handler = nifty_cooja:start("…", "…xmac.csc", []),
      Mote_IDs = add_motes(Handler, Motes),
      ok = nifty_cooja:simulation_step(Handler, 120000),
      DutyCycle = duty_cycle(Handler, Mote_IDs),
      R = check_duty_cycling(DutyCycle, 0.1),
```

- Retrieve and check the duty cycle

UPPSALA UNIVERSITET

Property

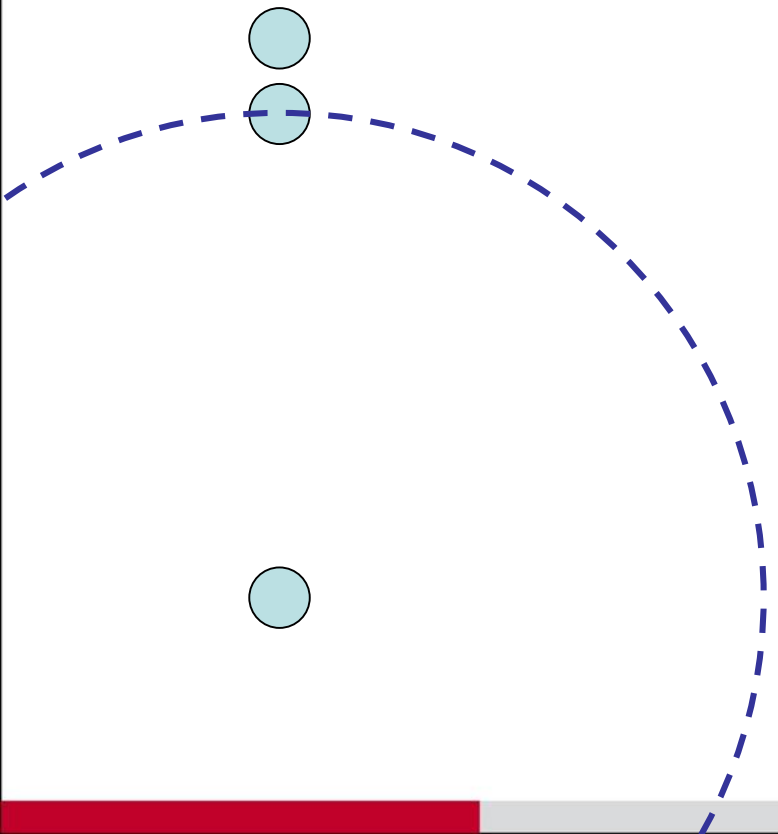
```
prop_xmac() ->
  ?FORALL(Motes, motes(),
    begin
      Handler = nifty_cooja:start("…", "…xmac.csc", []),
      Mote_IDs = add_motes(Handler, Motes),
      ok = nifty_cooja:simulation_step(Handler, 120000),
      DutyCycle = duty_cycle(Handler, Mote_IDs),
      R = check_duty_cycling(DutyCycle, 0.1),
      ok = nifty_cooja:exit(),
      R
    end).
```

- cleanup

UPPSALA UNIVERSITET

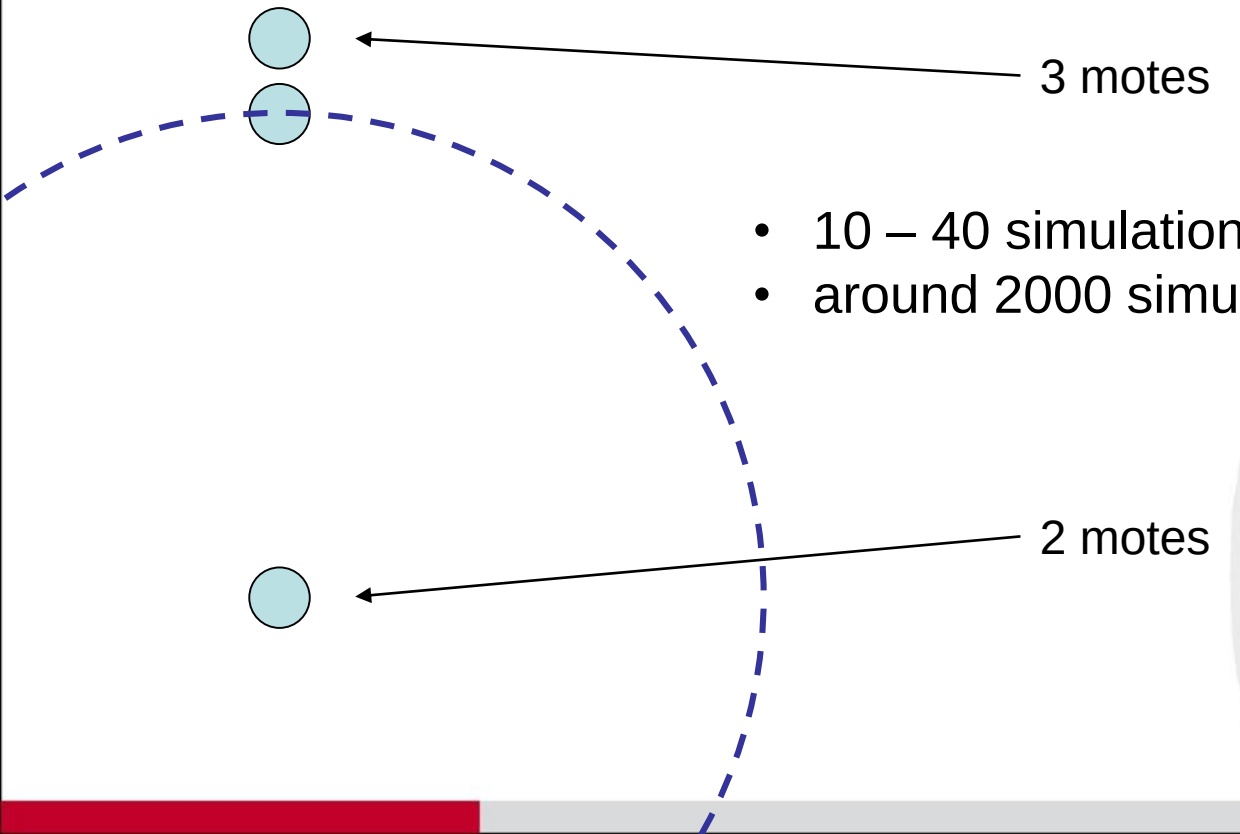
Results

- Counterexample with around 15 motes which can be shrunk down to 6 motes



Results

- Counterexample with 15 motes which was shrunk down to 6 motes



- 10 – 40 simulations to find counterexample
- around 2000 simulations for shrinking

UPPSALA UNIVERSITET

Results

- PropEr shrinks based on the used generators

- The Network topology is infered by the placement
- Shrinking influences the network topology only indirect

- Generators that produce graphs perform much better

UPPSALA UNIVERSITET

Testing the Encoder and Decoder of a Protocol on a Sensor Node

- Functions:

```
extern int encode(char*, char**);
extern int decode(char*, char**);
```

- Does decoding an encoded message yield the original message?
- Test it!

UPPSALA UNIVERSITET

Nifty for ContikiOS

- Contiki OS (http://www.contiki-os.org/) interface generator
- http://parapluu.github.io/nifty-contiki/

UPPSALA UNIVERSITET

Nifty for ContikiOS

```
0 = contiki_protocol:encode(Handler, 1, Message, Encoded)
```

Simulation Handler

Mote ID

UPPSALA UNIVERSITET

Demo

UPPSALA UNIVERSITET

Contiki's Socket API

- C-API for handling TCP sockets in Contiki
- Non-Blocking (return values over an event handler)

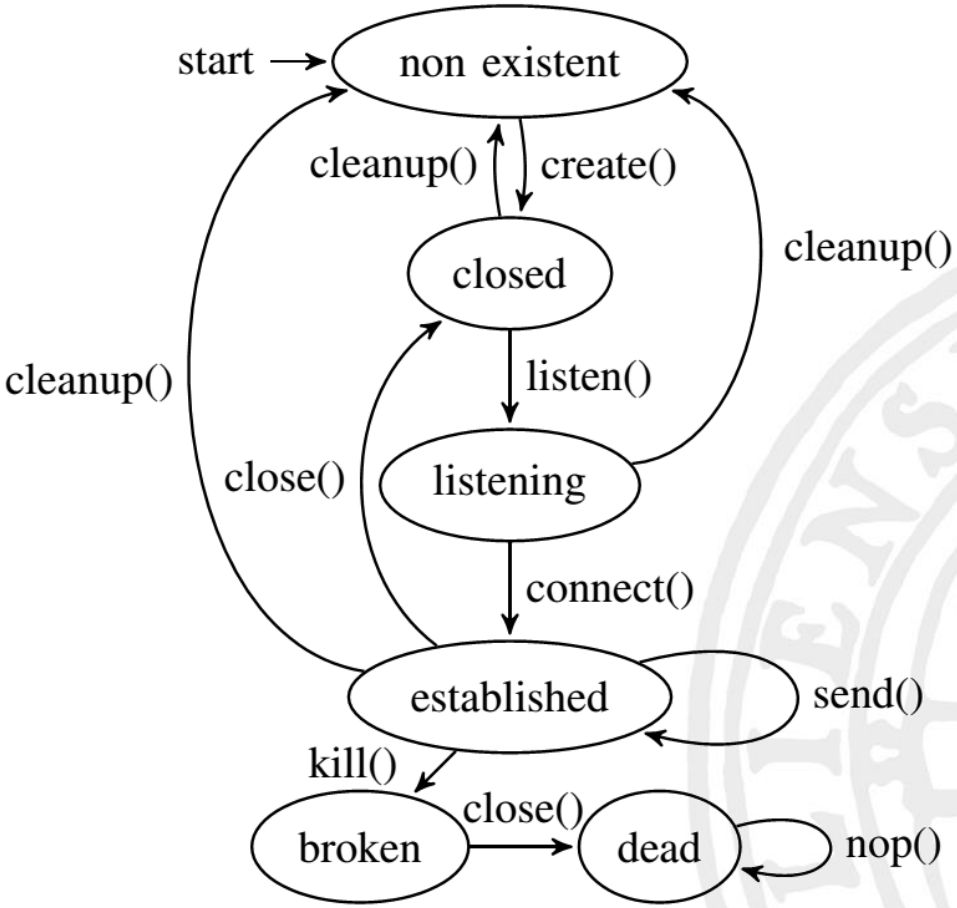
- Test:
  - Are the correct events triggered?

UPPSALA UNIVERSITET

Input

- Input:
  - List of function calls to the socket interface
- A complete random order of the function calls makes not much sense.
- We use an Finite State Machine to restrict the possible combinations of calls.

FSM for operations on 2 Sockets



UPPSALA UNIVERSITET

Results

1. Reception of an empty message after connect() that was never sent
2. Double ”closed” event on socket that was remotely closed
3. Missing ”closed” event after a sequence of 14 commands, which was shrunk to 8 commands

UPPSALA UNIVERSITET

Results

```
create  -> listen  -> connect ->
cleanup -> create  -> listen  ->
connect -> close (on socket that
                  listened)
```

- Any change on the sequence will make the bug not show