Property-Based Testing of Sensor Networks

Andreas Löscher, Konstantinos Sagonas, and Thiemo Voigt

Department of Information Technology, Uppsala University, Sweden

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

Contribution

- Extension of Property Based Testing (PBT) to Sensor Networks
- PBT Framework
- Case Studies:
  - XMAC duty-cycling
  - Contiki TCP Socket API

Testing an Encoder and a Decoder of a Protocol Implementation

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

Some test cases

$$assert\left(decode\left(encode("")\right) = ""\right)$$

$$assert\left(decode\left(encode("Hello\ World")\right) = "Hello\ World"\right)$$

- $assert\left(decode\left(encode("TestTestTest")\right) = "TestTestTest\right)$

…

Are those tests good?

- Look at code
- code coverage tools
- Write more tests
- Write more tests

Property-Based Testing

- Methodology for Software Testing
- Examples:
  - Quickcheck
  - PropEr
  - ScalaCheck
- We extend PBT to Sensor Networks

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

Example

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

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

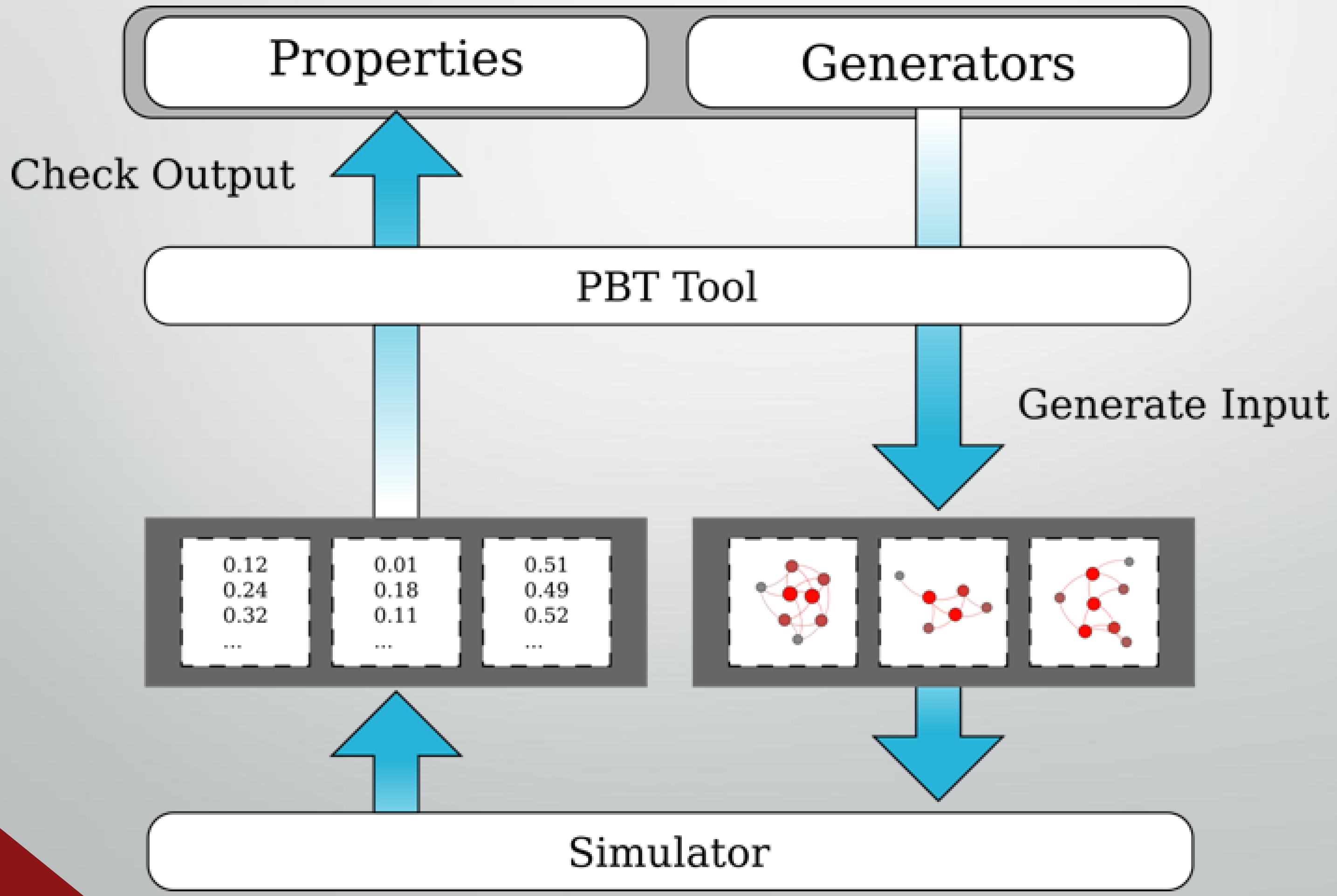
Testing

```
Eshell V6.3  (abort with ^G)
1> proper:quickcheck(protocol_test:prop_encode_decode()).
.............................................................
..!
Failed: After 64 test(s).
[45,80,58,119,94,62,118,71,71,119,114,123,75,67,62,84,99,60,
61,86,67]
Shrinking ...................(19 time(s))
[32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32]
false
```

Testing Sensor Networks

- Distributed Systems
  - Network Topologies
  - Heterogeneous Hardware
- Functional and Non-Functional Properties
  - Energy Consumption
  - Timing

Framework


Properties
Generators
Check Output
PBT Tool
Generate Input
0.12
0.24
0.32
...
0.01
0.18
0.11
...
0.51
0.49
0.52
...
Simulator

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%?

Property

```
prop_duty_cycle_below_threshold() ->
  ?FORALL(Motes, configuration(),
```

- Generates a random configuration of motes
- Motes:
  - Position (x,y)
  - Mote Id
  - Type (Server/Client)

Property

```
prop_duty_cycle_below_threshold() ->
  ?FORALL(Motes, configuration(),
     begin
       setup(),
       {running, Handler} = nifty_cooja:state(),
       Mote_IDs = add_motes(Handler, Motes),
```

- Start and initialize the simulation

Property

```
prop_duty_cycle_below_threshold() ->
  ?FORALL(Motes, configuration(),
     begin
       setup(),
       {running, Handler} = nifty_cooja:state(),
       Mote_IDs = add_motes(Handler, Motes),
       SimTime = 120 * 1000,
       nifty_cooja:simulation_step(Handler, SimTime),
```

- Run the simulation

Property

```
prop_duty_cycle_below_threshold() ->
  ?FORALL(Motes, configuration(),
     begin
       setup(),
       {running, Handler} = nifty_cooja:state(),
       Mote_IDs = add_motes(Handler, Motes),
       SimTime = 120 * 1000,
       nifty_cooja:simulation_step(Handler, SimTime),
       MaxDutyCycle = max_duty_cycle(Handler, Mote_IDs),
```

- Calculate the maximum of the duty-cycle of the motes

Property

```
prop_duty_cycle_below_threshold() ->
  ?FORALL(Motes, configuration(),
     begin
       setup(),
       {running, Handler} = nifty_cooja:state(),
       Mote_IDs = add_motes(Handler, Motes),
       SimTime = 120 * 1000,
       nifty_cooja:simulation_step(Handler, SimTime),
       MaxDutyCycle = max_duty_cycle(Handler, Mote_IDs),
       MaxDutyCycle < 0.1
     end).
```

- Check if the duty-cycle is below 10%

Results

1. Counterexample with 15 motes which was shrunk down to 6 motes

What about ContikiMac?

2. The same test with ContikiMac; no Counterexample after 1000 tests

Contiki's Socket API

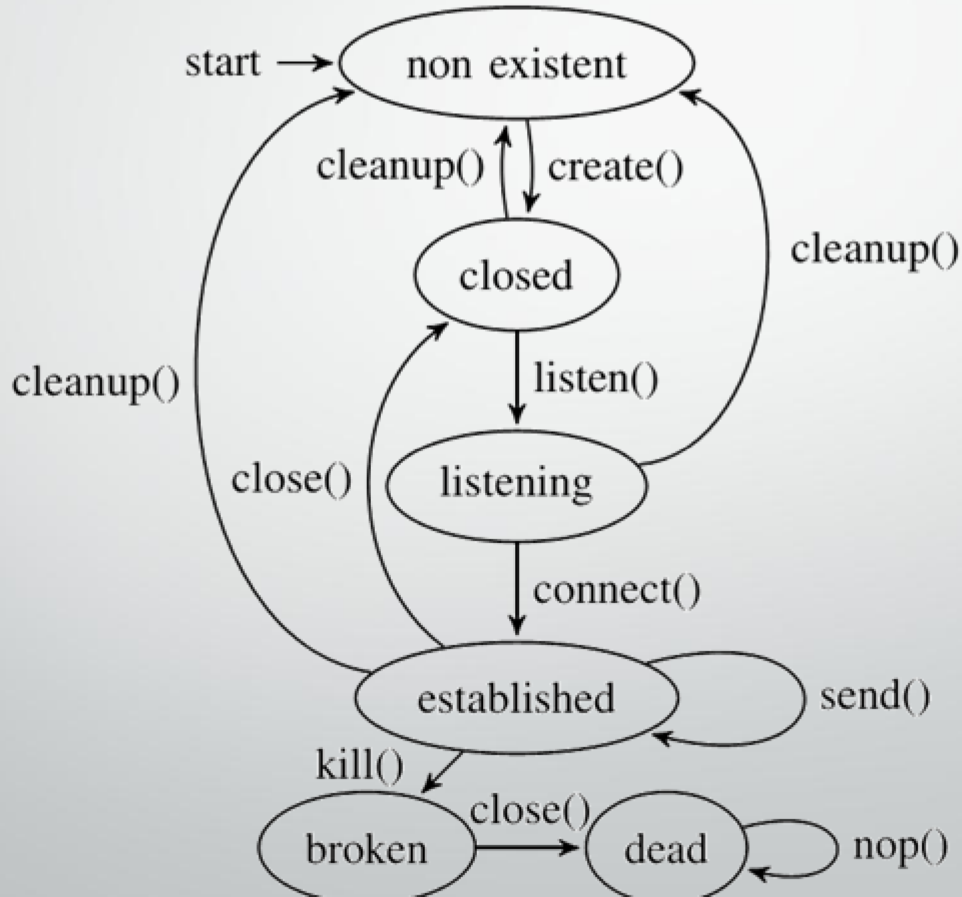
- C-API for handling TCP sockets in Contiki
- Non-Blocking (return values over an event handler)

- Test:
  - Are the correct events triggered?

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



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

Results

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

- Any change in the sequence will make the bug not show
- Hard to find for a human tester

Conclusion

- Property-Based Testing is an effective way to test sensor networks.
- We provide a framework that can be applied to a wide variety of sensor network applications.
- Can already be used to find real, hard-to-find bugs in sensor network applications.