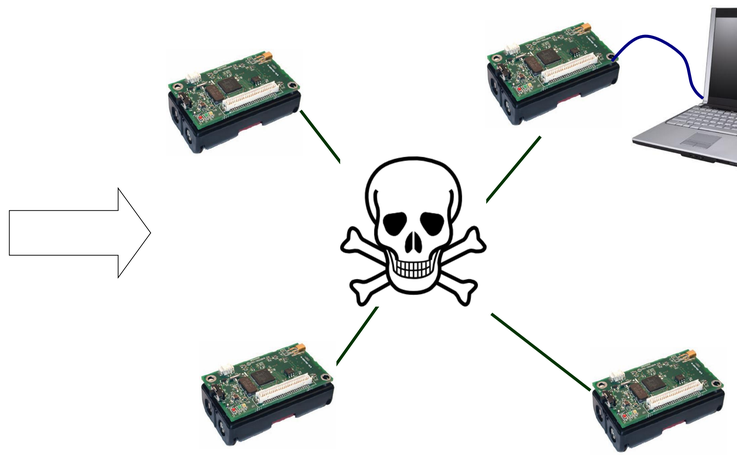
T-Check: Bug Finding for Sensor Networks

THE UNIVERSITY OF UTAH

Peng Li, John Regehr
School of Computing, University of Utah
{peterlee, regehr}@cs.utah.edu

Why T-Check?



1. Where are bugs located??
2. How are causes pinpointed??

1. Physically unreachable nodes
2. Error trace unable to be reproduced

A tool to find safety and liveness errors in sensor network application running on TinyOS before deployment! (Accepted in IPSN 2010)

Safety & Liveness Property

▪ A ***safety*** property is true if something bad never happens. If any state in an execution violates the safety property, the entire execution violates that property.

▪ A ***liveness*** property holds if something good will eventually happen. An execution satisfies a liveness property if the execution will encounter a *live* state in finite time.

Type	Property
Safety	The current buffer, if unlocked, should successfully receive an incoming packet
Liveness	Eventually, each of two buffers becomes unlocked
Liveness	Eventually, there is no loop in the routing tree
Liveness	Eventually, all nodes that have a path to a sink become part of SOME collection tree

Table 1 . Part of TinyOS-2.x properties we checked

```
command error_t DipDecision.send() {
   dip_msg_t* dmsg;
   dmsg = (dip_msg_t*) call  SummarySend.getPayloadPtr();
   … Assert(dmsg != NULL) …
   // BUG -- dsmg can be NULL here
   dmsg->type = ID_DIP_SUMMARY;
   ... code omitted ...
}
```

Typical Safety Property

Fig 1. Safety property illustrated via a DIP bug

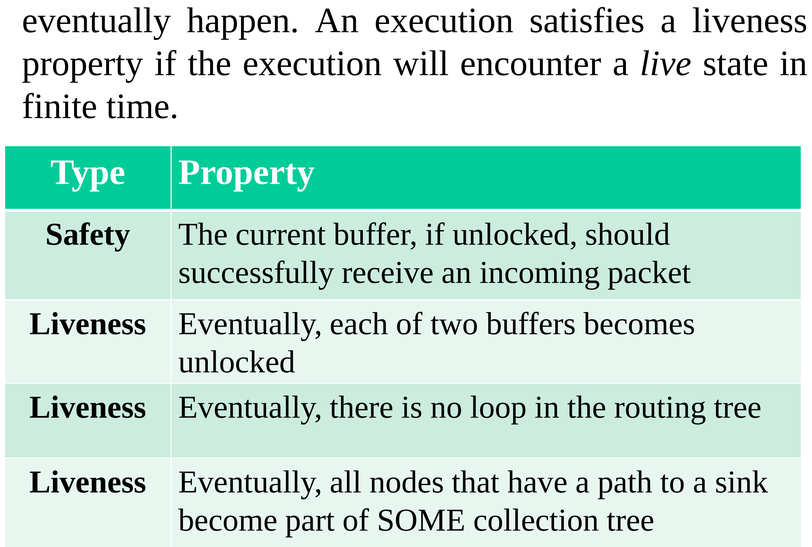


Fig2. Liveness property illustrated via CTP

T-Check

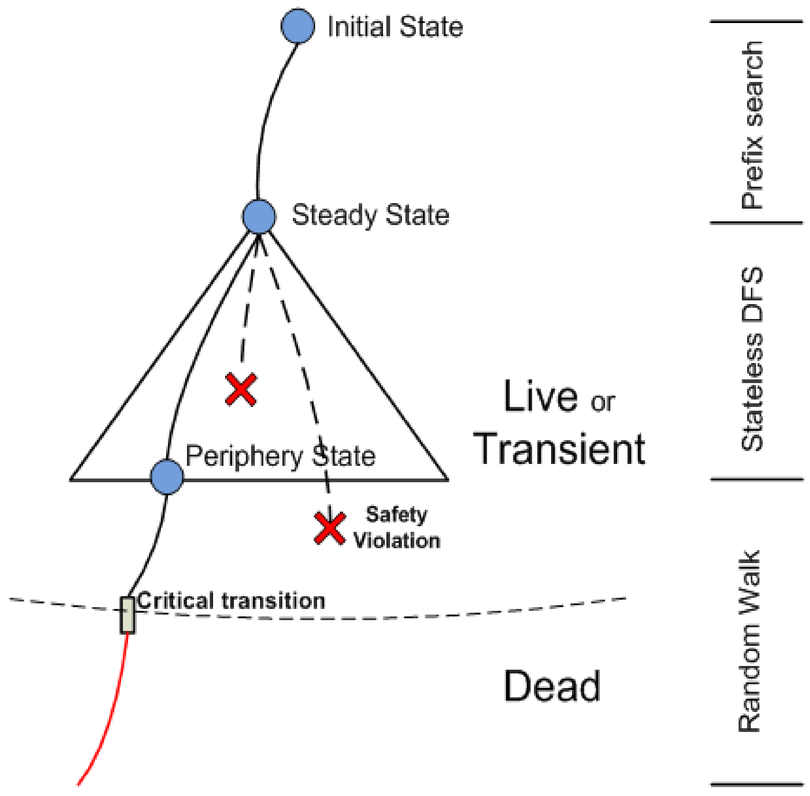


Fig3. T-Check execution procedure

1. Prefix random execution to get sensornet into a steady state (**Optional**)
2. Depth-bounded explicit state model checking to exhaustively explore the state space up to some depth
3. Long-way random walks to find additional safety violations and verify that potential liveness violations are transient or dead.
4. Additional random walks to shorten the safety error trace dumped in random walk phase
5. Additional random walks to find the critical transitions

➢(**Safety violation**) T-Check dumps a pruned execution trace leading to the violation.
➢(**Liveness violation**) T-Check searches for the critical transition and dumps an execution trace leading to this transition.

Evaluation

12 previously unknown bugs have been caught, some of them in core services and in applications that have been used for several years.

Component	Safety	Liveness
MultihopOscilloscopeC	1	0
LinkEstimatorP	0	1
SerialDispatcherP	1	1
DipSummaryP	2	0
DipVersionP	1	0
MHPacketM	1	0
DhvSummaryP	1	0
DhvVBitP	2	0
DhvHSumP	1	0

Table 2 . Summary of bugs found by T-Check

```
typedef struct oscilloscope {
   ... code omitted ...
   uint16_t readings[NREADINGS];
} oscilloscope_t;
oscilloscope_t local;

uint8_t reading;

event void timer.fired() {
   if (reading == NREADINGS) {
.      ... code omitted ...
      reading = 0;
   }
   if (call Read.read() != SUCCESS)
         fatal_problem();
}

event void Read.readDone(error_t result, uint16_t data) {
    ... code omitted ...
   // BUG when reading >= NREADINGS
   local.readings[reading++] = data;
}
```

Fig 4. Code for a bug in MultihopOscilloscope

http://www.cs.utah.edu/~peterlee/tcheck