PRISM:

A Tool For Stochastic Model Checking

Dave Parker

University of Birmingham

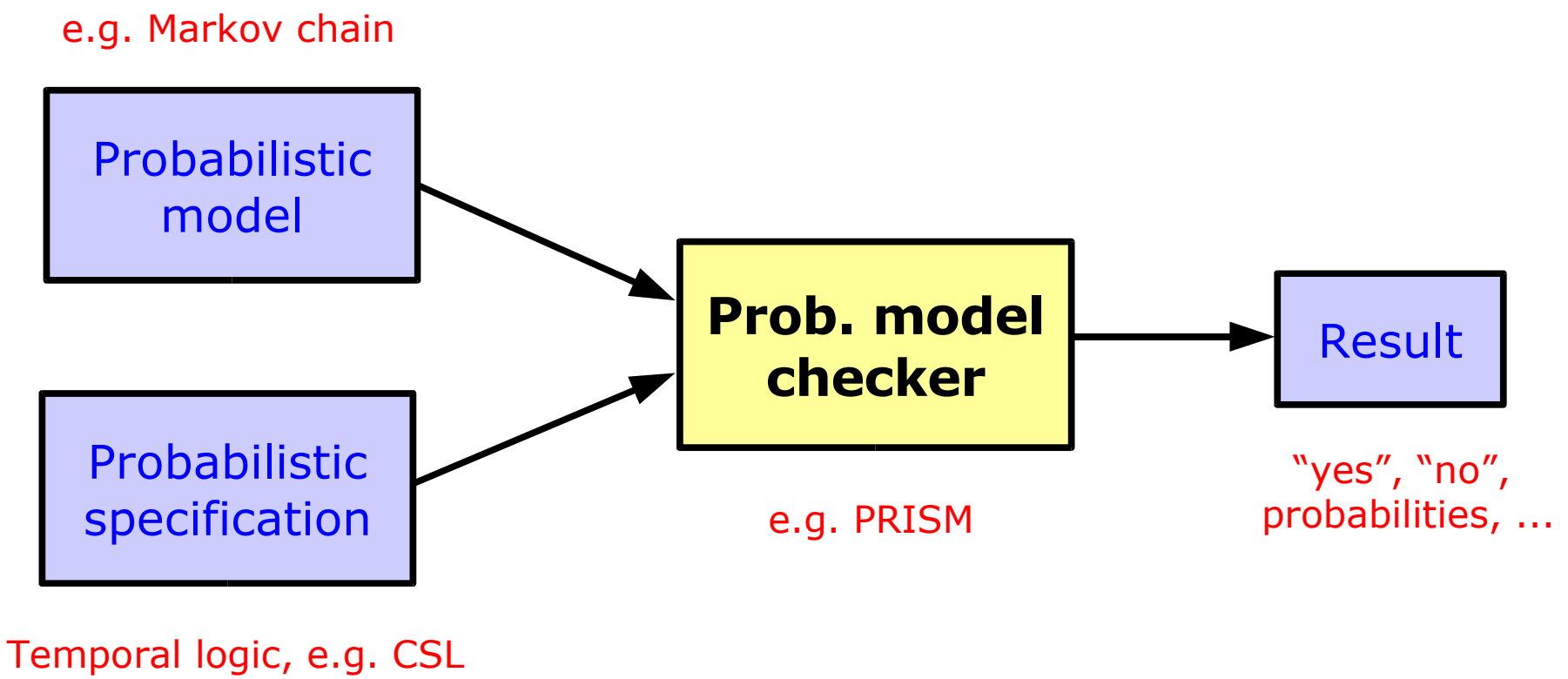
ARTIST2 Summer School, October 2005

Overview

- Stochastic/probabilistic model checking
  - overview
- Tool support: PRISM
  - modelling language
  - property specification
  - functionality, efficiency
- Case studies
  - self-stabilisation: Herman's algorithm
  - Bluetooth device discovery
  - FireWire root contention

Probabilistic model checking

- Automatic formal verification of probabilistic systems
  - exhaustive exploration/analysis of model



Probabilistic Models

- Discrete-time Markov chains (DTMCs)
  - time modelled in discrete steps; discrete probabilities
- Continuous-time Markov chains (CTMCs)
  - continuous (real-valued) model of time: exponential distributions
- Markov decision processes (MDPs)
  - discrete time-steps, discrete probabilities
  - both probabilistic and nondeterministic behaviour
  - e.g. parallel composition of concurrent stochastic processes
- Probabilistic timed automata (PTAs)
  - extension of timed automata
  - discrete probabilities, nondeterminism, real-time clocks
  - in some cases, can transform to MDPs ("digital clocks")

Probabilistic model checking

+ wide range of quantitative measures
+ exact answers computed
+ fully automatic process
  - model construction + numerical solution
– no counterexamples
  - but can identify patterns, trends, anomalies in quantitative results
+ exhaustive analysis, good for 'corner cases', e.g.
  - all possible initial configurations/model parameter values
  - all possible process schedulings (nondeterminism)
– state space explosion: time and memory constraints
+ efficient algorithms and implementations

Probabilistic model checkers

- PRISM – DTMCs, CTMCs, MDPs
- ETMCC – DTMCs, CTMCs, action-based logics
- Also:
  - RAPTURE, APNN-Toolbox, Prob-USM, ...
- Simulation-based probabilistic model checking
  - APMC, Ymer (formerly ProVer)
- Multiple formalism/tool solutions
  - CADP, Möbius
- Non-probabilistic model checking of probabilistic models
  - TwoTowers, SMART, ...

PRISM

- PRISM: Probabilistic model checker
  - developed at the University of Birmingham, approx. 7 years
  - free, open source: Linux, Unix, Mac OS X, Windows
  - www.cs.bham.ac.uk/~dxp/prism

- Models: DTMCs, CTMCs, MDPs + costs/rewards
- Properties: PCTL, CSL + extensions + costs/rewards

- Features:
  - high-level modelling language
  - wide range of model analysis methods
  - graphical user interface
  - efficient implementation

PRISM modelling language

- Simple, state-based language for DTMCs/CTMCs/MDPs
  - based on Reactive Modules [Alur/Henzinger]
- Modules (system components, composed in parallel)
- Variables (finite-valued, local or global)
- Guarded commands (labelled with probabilities/rates)
- Action labellings (synchronisation between modules)
  - also used for process algebra-style operations

$$[send]\ (s=2) \rightarrow p_{loss} : (s'=3)\&(lost'=lost+1) + (1-p_{loss}) : (s'=4);$$

action guard probability update probability update

PRISM language example (fragment)

```
const int MIN_SENSORS = 2;
const double lambda_p = 1/(365*24*60*60); // 1 year
const double lambda_s = 1/(30*24*60*60); // 1 month
const double delta_f = 1/(24*60*60); // 1 day
const double delta_r = 1/30; // 30 secs

module sensors
    s : [0..3] init 3; // number of sensors working
    [] s>1 -> s*lambda_s : (s'=s-1); // failure of a single sensor
endmodule

module proci // (takes data from sensors and passes onto main processor)
    i : [0..2] init 2; // 2=ok, 1=transient fault, 0=failed
    [] i>0 & s>=MIN_SENSORS -> lambda_p : (i'=0); // failure of processor
    [] i=2 & s>=MIN_SENSORS -> delta_f : (i'=1); // transient fault
    [reboot] i=1 & s>=MIN_SENSORS -> delta_r : (i'=2); // reboot after transient fault
endmodule
```

PRISM – Property specifications

- Formulae in probabilistic extensions of temporal logic
  - PCTL for DTMCs/MDPs, CSL for CTMCs
- Examples:
  - P<0.001 [F shutdown] - "shutdown eventually occurs with probability at most 0.001"
  - P<0.2 [F[t,t] (deliv_rate < min)] "the probability that the current packet delivery rate has dropped below minimum at time t is less than 0.2"
  - P≥0.95 [!repair U≤200 done] - "with probability 0.95 or greater, the process will successfully complete within 200 hours and without requiring any repairs"
  - S>0.75 [num_sensors ≥ min] - "in the long-run, the probability that an adequate number of sensors are operational is greater than 0.75"

Computing actual values

- Determine actual probabilities
  - P=? [F shutdown] - “what is the probability of shutdown eventually occurring?”
  - S=? [num_sensors ≥ 4] - “what is the long-run probability that 4 sensors are operational?”
- Probability in a specific state of the model
  - P=? [F shutdown {x=11 & y=17}]
- Best/worst-case probabilities
  - P=? [F shutdown {“init”}{max}]
  - Pmin=? [F shutdown]
- Experiments – ranges of model/property parameters
  - P=? [F≤T error] for N=1..5, T=1..100

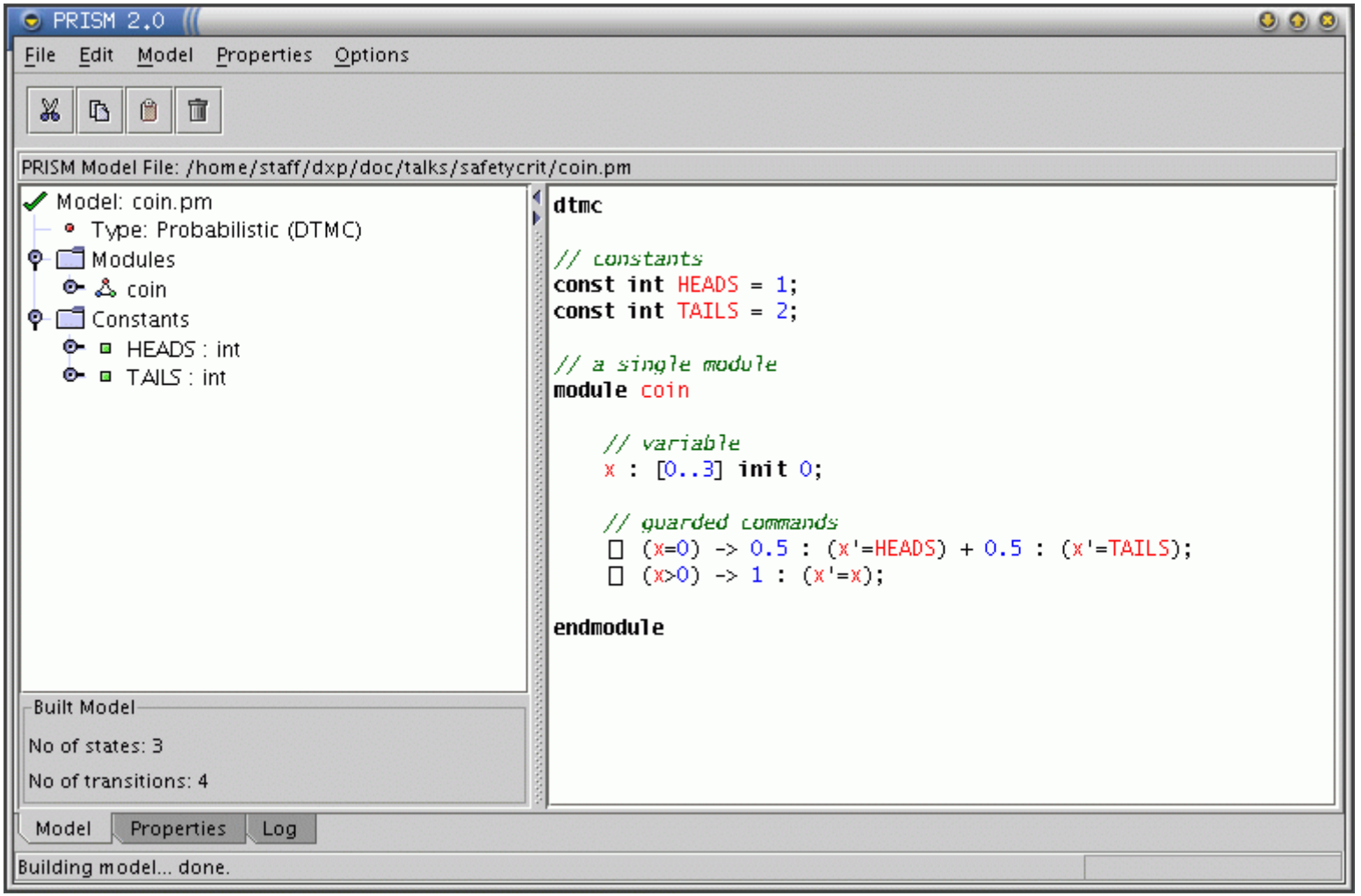
Cost- and reward-based properties

- Costs and rewards
  - real-valued measures assigned to states/transitions
- Instantaneous – assigned to states
  - current queue size, number of operational channels, ...
  - "what is the expected size of the message queue at time t?"
  - "what is the long-run expected size of the queue?"
- Cumulative – states or transitions
  - time, power consumption, messages lost, ...
  - "what is the expected power consumption during the first 2 hours of operation?"
  - "what is the expected time taken for the protocol to terminate?"

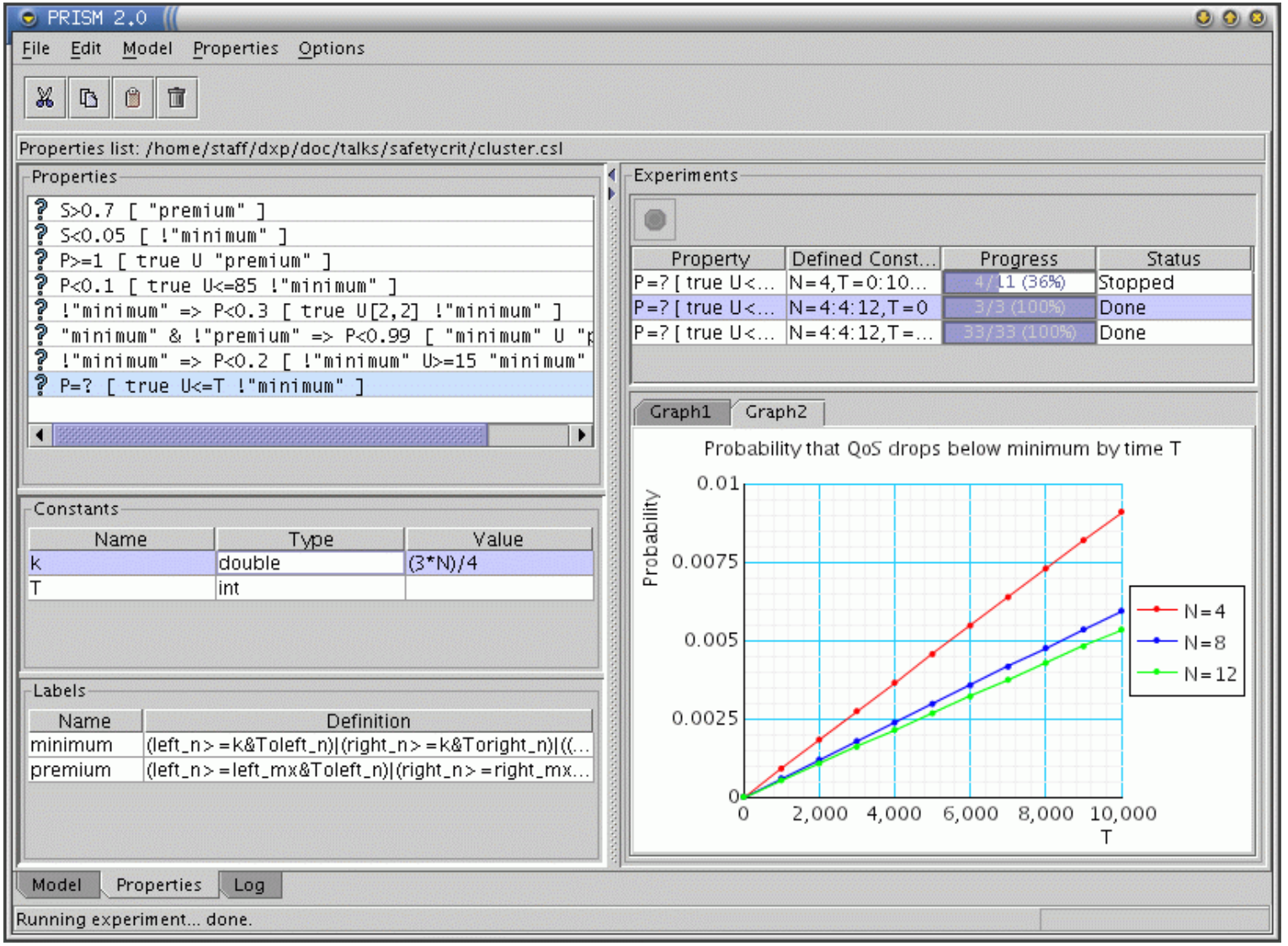
PRISM - Functionality

- Model construction
  - convert model description to DTMC, CTMC, MDP
  - compute reachable state space
- Model checking
  - graph-based computations: reachability, etc.
  - numerical solution: linear equation systems, optimisation problems (dynamic programming), ...
- Graphical user interface
  - PRISM language editor, experiments, graph plotting, ...
- Discrete-event simulation (ongoing work)
  - manual (debugging) or automatic (Monte Carlo sampling)

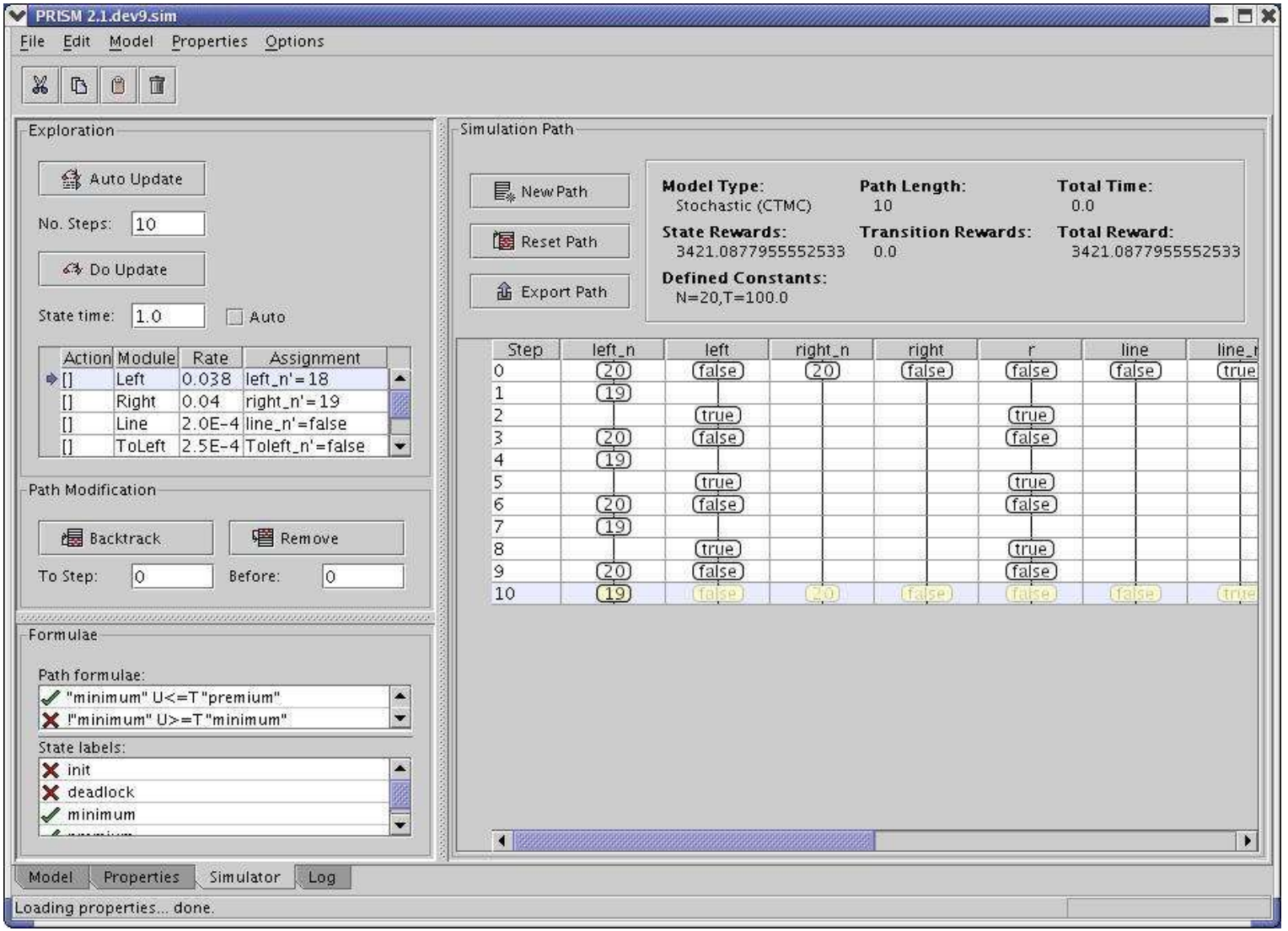
PRISM - Screenshots



PRISM - Screenshots



PRISM - Screenshots



PRISM - Efficiency

- State space explosion
  - models of real-life systems typically huge
- Symbolic probabilistic model checking
  - data structures based on binary decision diagrams (BDDs)
  - compact storage: exploit model structure and regularity
- PRISM: multiple computation engines
  - MTBDDs (BDD extension): allows storage/analysis of extremely large models (given structure/reguarity), numerical computation can blow up
  - sparse matrices: fastest solution for smaller models, prohibitive memory consumption for larger models
  - hybrid: combine MTBDD storage with explicit storage, ten-fold increase in analysable model size

PRISM - Case studies

- Wide range of application domains
  - communication and multimedia protocols
  - security protocols – anonymity, contract signing, ...
  - randomised distributed algorithms
  - dynamic power management
  - biological processes – cell cycle pathways, ...
  - performance/reliability of manufacturing systems, computer networks, NAND multiplexing, ...

- www.cs.bham.ac.uk/~dxp/prism/casestudies

Case study - Self-stabilisation

- Self-stabilising protocol for a network of processes
  - starts from possibly illegal start state
  - returns to a legal (stable) state
    - without any outside intervention
    - within some finite number of steps
- Herman's self-stabilising protocol
  - directed synchronous ring of N processes (N odd)
  - each process can possess a token
    - modelled by: N Boolean variables x_1 ... x_n
    - process i has a token if $x_i = x_{(i+1) \bmod N}$
  - illegal states: more than one process with a token
  - stable states: exactly one process with a token

Herman's self-stabilising protocol

- Basic algorithm
  - at each step, if process i has a token, it sets x_i randomly
  - otherwise, sets x_i to x_{i+1}
  - effectively: randomly pass tokens on, merge multiple tokens
- Modelling in PRISM
  - discrete probabilities, discrete time-steps, synchronous processes
  - use discrete-time Markov chain (DTMC)

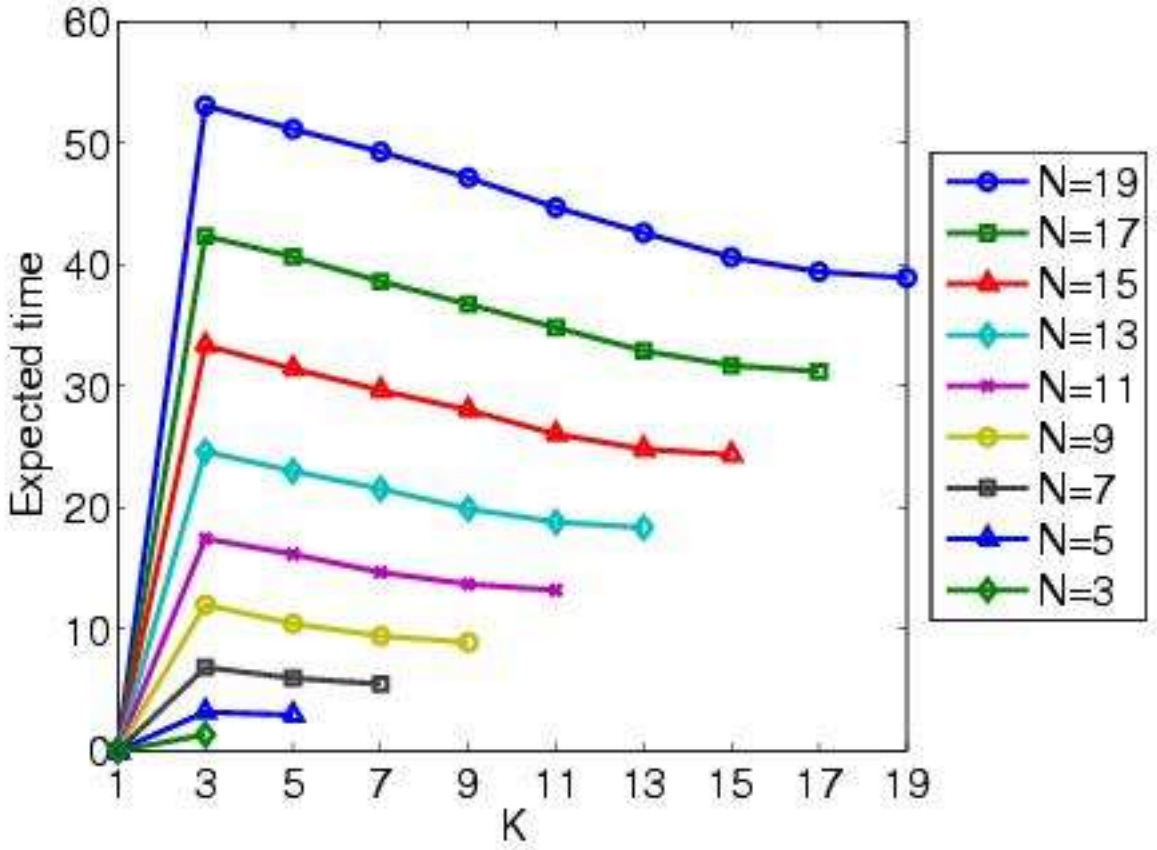
```
module process1
    x1 : bool;
    [step]    x1=x2   -> 0.5 : (x1'=false) + 0.5 : (x1'=true);
    [step]   !(x1=x2) -> (x1'=x2);
endmodule

module process2  = process1 [x1=x2, x2=x3] endmodule
...
module processN = process1 [x1=xN, x2=x1] endmodule
```

Herman's self-stabilising protocol

- Properties of interest:
  - "from any state, stable state reached with probability 1"
  - "expected time to reach a stable state"
- Labels:
  - count : (x1=x2?1:0) + (x2=x3?1:0) + ... + (xN=x1?1:0)
  - "stable" : count=1
- P≥1 [F "stable" {"init"}{min}]
  - "min probability of reaching a stable state is 1"
  - TRUE

Results: Herman's protocol

- R=? [F "stable" {count=K}{max}]
  - "worst-case expected number of steps to reach a stable state for initial configurations with K tokens"



Results support unproven conjecture [McIver,Morgan'04] that worst case is always for 3 tokens

Case study: Bluetooth

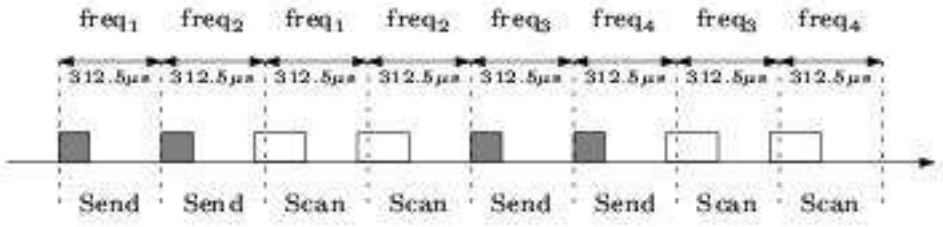
- Bluetooth: short-range low-power wireless protocol
  - widely available in phones, PDAs, laptops, ...
  - personal area networks (PANs)
  - open standard, specification freely available
- Uses frequency hopping scheme
  - to avoid interference (uses unregulated 2.4GHz band)
  - pseudo-random selection over 32 of 79 frequencies
- Network formation
  - piconets (1 master, up to 7 slaves)
  - self-configuring: devices discover themselves

Bluetooth device discovery

- States of a Bluetooth device:
  - Standby: default operational state
  - Inquiry: device discovery
    - master looks for devices, slaves listens for master
  - Page: establish connection - synchronise clocks, etc.
  - Connected: device ready to communicate in a piconet
- Device discovery
  - manadatory first step before any communication possible
  - "page" reuses information from "inquiry" so is much faster
  - power consumption much higher for "page"
  - performance crucial

Master (sender) behaviour

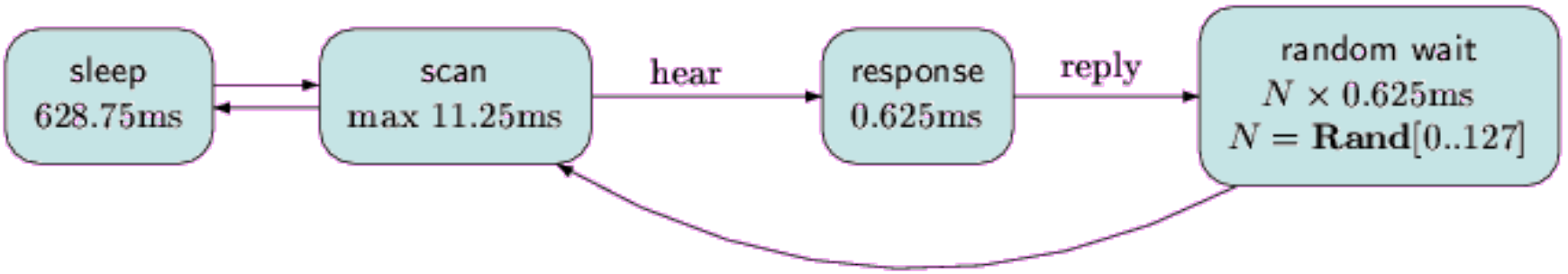
- 28 bit free-running clock CLK, ticks every 312.5µs
- Frequency hopping sequence determined by clock:
  - freq = [CLK_{16-12}+k+ ($CLK_{4-2,0}$- CLK_{16-12}) mod 16] mod 32
  - 2 trains of 16 frequencies (determined by offset k), 128 times each, swap between every 2.56s
- Broadcasts inquiry packets on two consecutive frequencies, then listens on the same two



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	16

Slave (receiver) behaviour

- Listens (scans) on frequencies for inquiry packets
  - must listen on right frequency at right time
  - cycles through frequency sequence at much slower speed (every 1.28s)



- On hearing packet, pause, send reply and then wait for a random delay before listening for subsequent packets
  - avoid repeated collisions with other slaves

Bluetooth device discovery

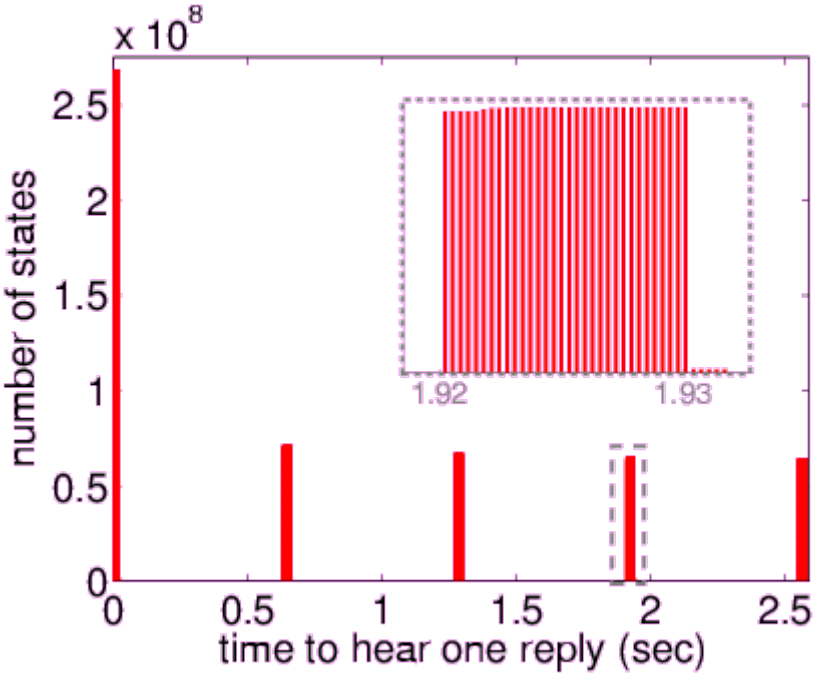
- Modelling in PRISM
  - model one sender and one receiver
  - synchronous (clock speed defined by Bluetooth spec)
  - randomised behaviour – use DTMC
  - model at lowest-level (one clock-tick = one transition)
  - use real values for delays, etc. from Bluetooth spec
- Modelling challenges
  - complex interaction between sender/receiver
  - combination of short/long time-scales – cannot scale down
  - sender/receiver not initially synchronised, state determined by 28 bit clock – huge number of possible initial configurations (17,179,869,184)

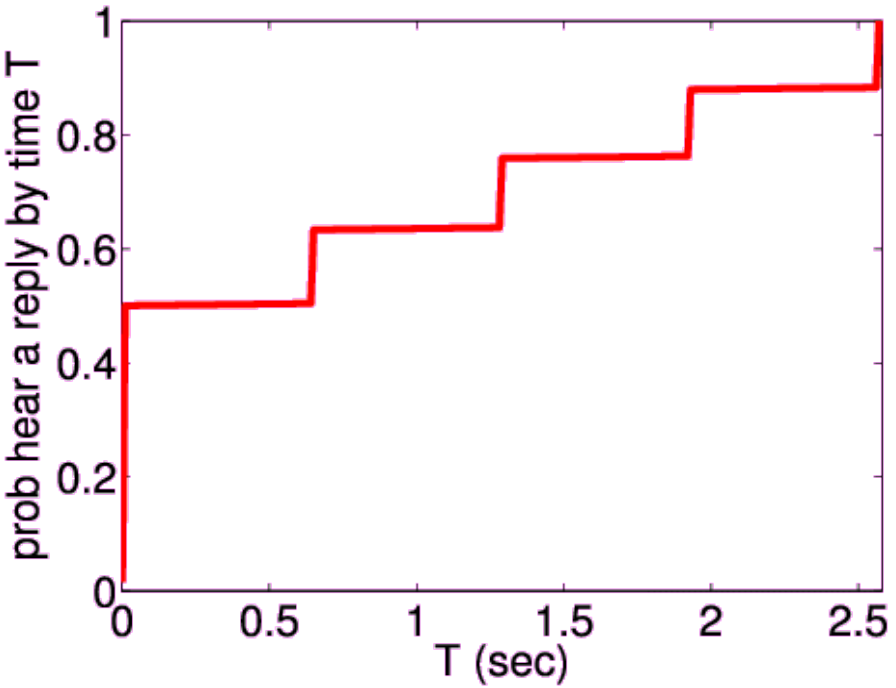
Bluetooth: Results

- Huge DTMC – initially, model checking infeasible
  - partition into 32 scenarios, i.e. 32 separate DTMCs
  - on average, approx. 3.4×10^9 states, 536,870,912 initial
  - can be built/analysed with PRISM's MTBDD engine

- Compute:
  - R=? [F replies=K {"init"}{max}]
  - "worst-case expected time to hear K replies over all possible initial configurations"
  - also look at:
    - how many initial states for each possible expected time
    - cumulative distribution function assuming equal probability for each initial state

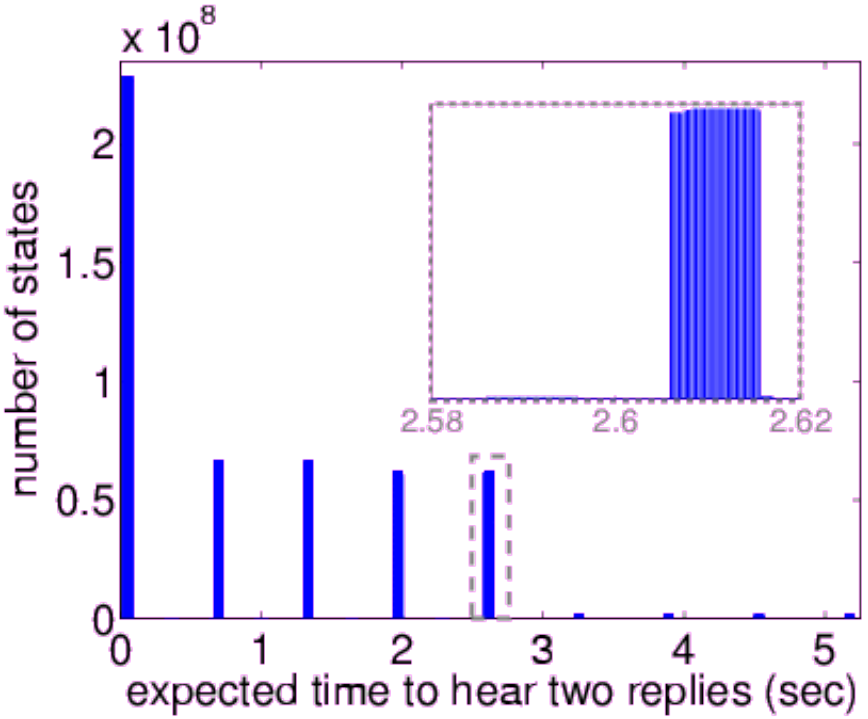
Bluetooth: Time to hear 1 reply

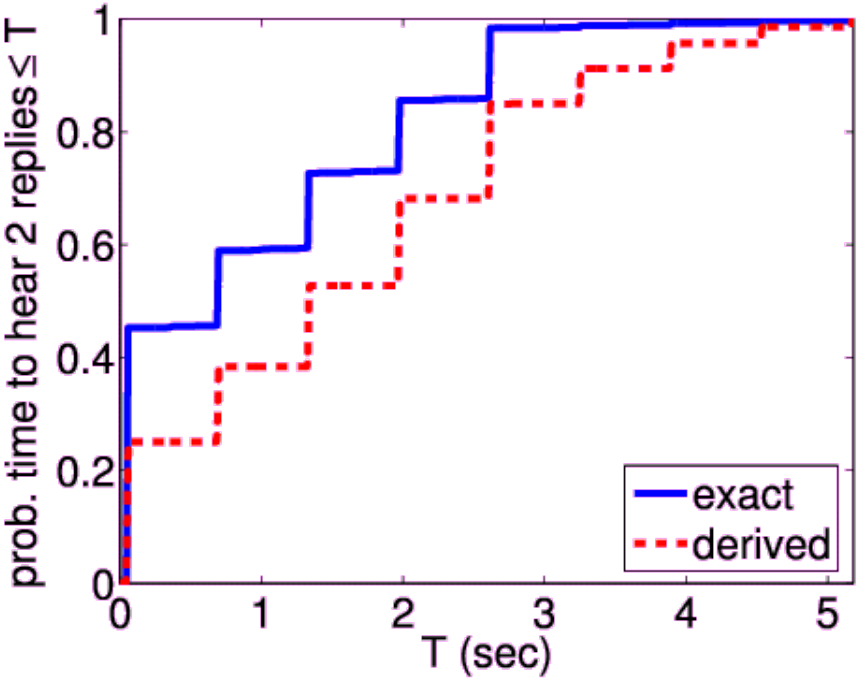




- worst-case expected time = 2.5716 sec
- in 921,600 possible initial states
- best-case = 635 μs

Bluetooth: Time to hear 2 replies





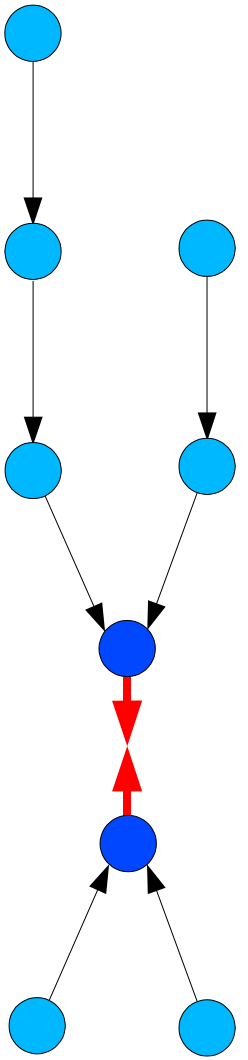
- worst-case expected time = 5.177 sec
- in 444 possible initial states
- compare actual CDF with derived version which assumes times to reply to first/second messages are independent

Bluetooth: Results

- Other results:
  - compare versions 1.2 and 1.1 of Bluetooth, confirm 1.1 slower
  - power consumption analysis
- Conclusions:
  - successful analysis of complex real-life model, actual parameters
  - exhaustive analysis: best-/worst-case values
    - can pinpoint scenarios which give rise to them
    - not possible with simulation approaches
  - model still realtively simple
    - consider multiple receivers?
    - combine with simulation?

Case study: FireWire root contention

- FireWire (IEEE 1394)
  - serial bus for networking multimedia devices
  - "hot-pluggable" - add/remove devices at any time
- Root contention protocol
  - leader election algorithm, when nodes join/leave
  - nodes send messages: "be my parent"
  - root contention: when nodes contend leadership
  - random choice: "fast"/"slow" delay before retry
- Properties of interest
  - time taken for leader election
  - effect of using biased coin - conjecture [Stoelinga]



FireWire - PRISM model

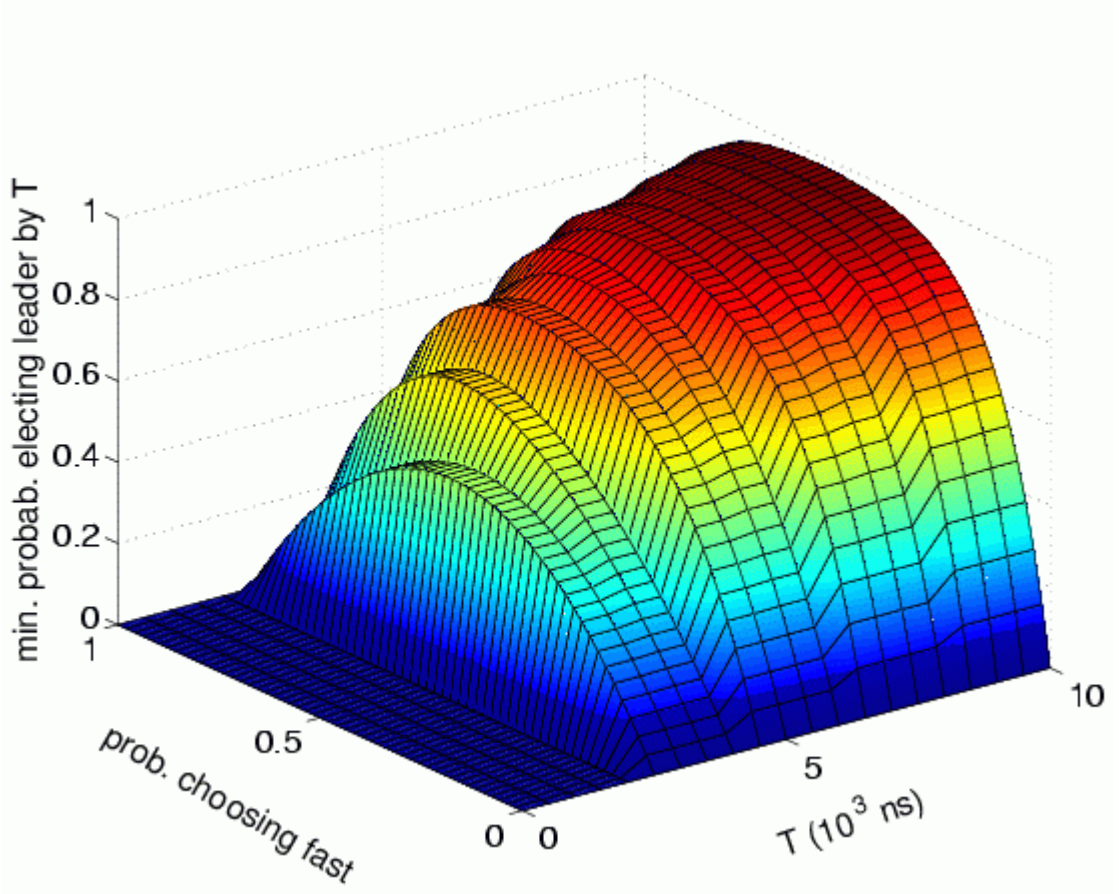
- Based on probabilistic timed automata (PTA) model of
  - [Stoelinga,Vaandrager'99], [Simons,Stoelinga'01]
  - infinite state (real-time)
  - [Kwiatkowska,Norman,Sproston'03]
  - abstraction: digital clocks - preserves properties of interest

- PRISM model: finite-state MDP
  - concurrency: messages between nodes and wires
  - underspecification of delays (upper/lower bounds)
  - probability: coin toss
  - max. model size: 170 million states
  - analysed using PRISM's MTBDD engine

FireWire - Model checking

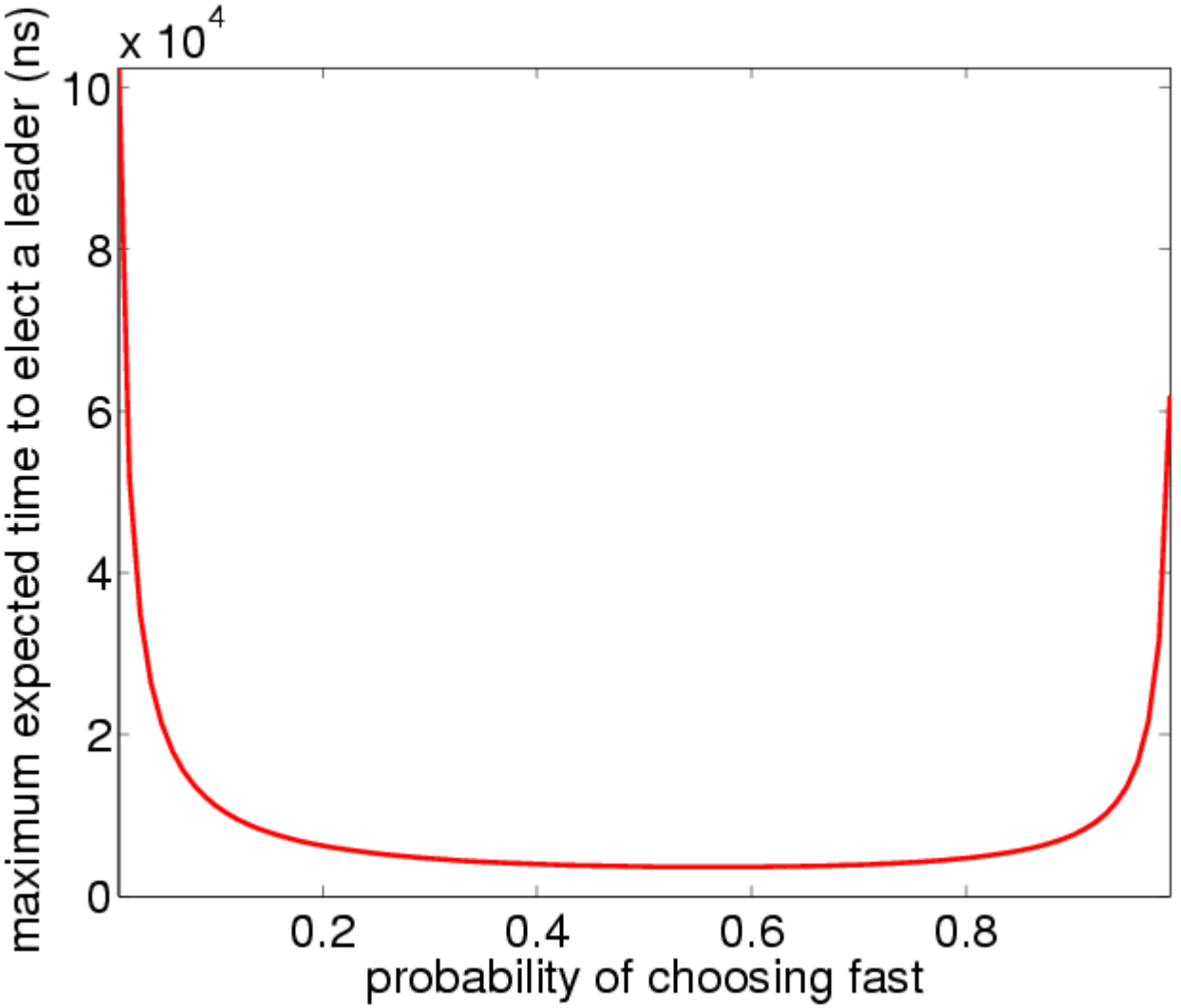
- "minimum probability that a leader is elected by time T"
  - add variable t to count elapsed time
  - Pmin=? [t≤T U “elected”]
  - vary: T, coin bias: probability of choosing "fast"
- "maximum expected time to elect a leader"
  - add timing costs
  - Rmax=? [F “elected”]
  - vary: coin bias

FireWire - Results



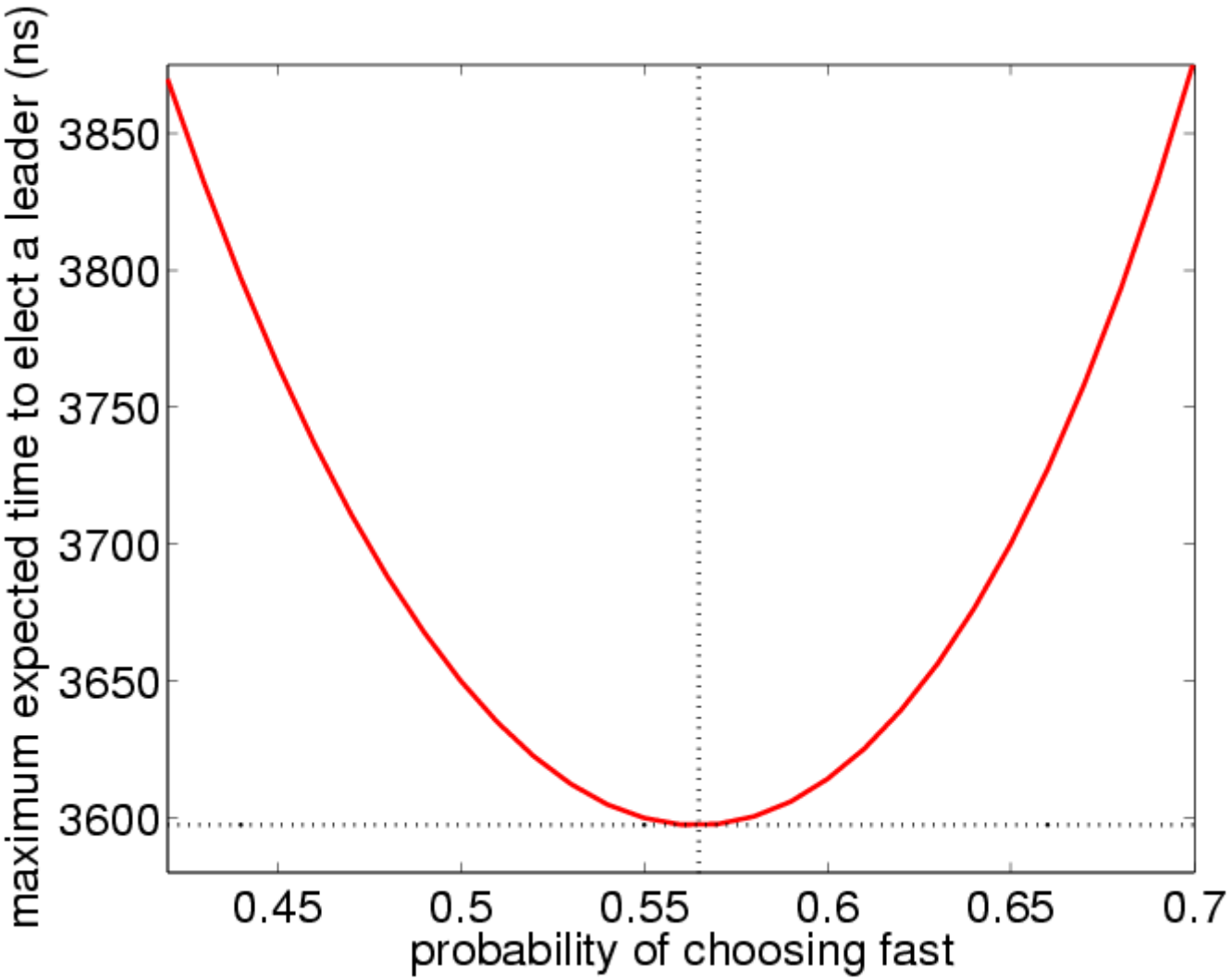
"minimum probability of electing leader by time T"

FireWire - Results



"maximum expected time to elect a leader"

FireWire - Results



"maximum expected time to elect a leader"

Biased coin is beneficial

Conclusions

- Probabilistic model checking + PRISM
  - formal analysis of probabilistic systems
    - automatic, exhaustive
  - wide range of models/properties

- Ongoing/future work:
  - PRISM functionality
    - discrete event simulation, graphical modelling language
    - native support for real-time, mobility
  - efficiency of probabilistic model checking
    - parallel, distributed, Grid-based computation
    - symmetry, abstraction, compositionality
  - PhD studentship available: www.cs.bham.ac.uk/~dxp/prism