Experiments on the PGM protocol

Anahita Akhavan, Marius Bozga, Yassine Lakhnech

VERIMAG

Marc Boyer, Ahmed Bouajjani

LIAFA

Outline

- Modeling
  - the protocol
  - the environment
- Verification
  - state-space generation
  - model checking
- Conclusions
- Future work

The protocol

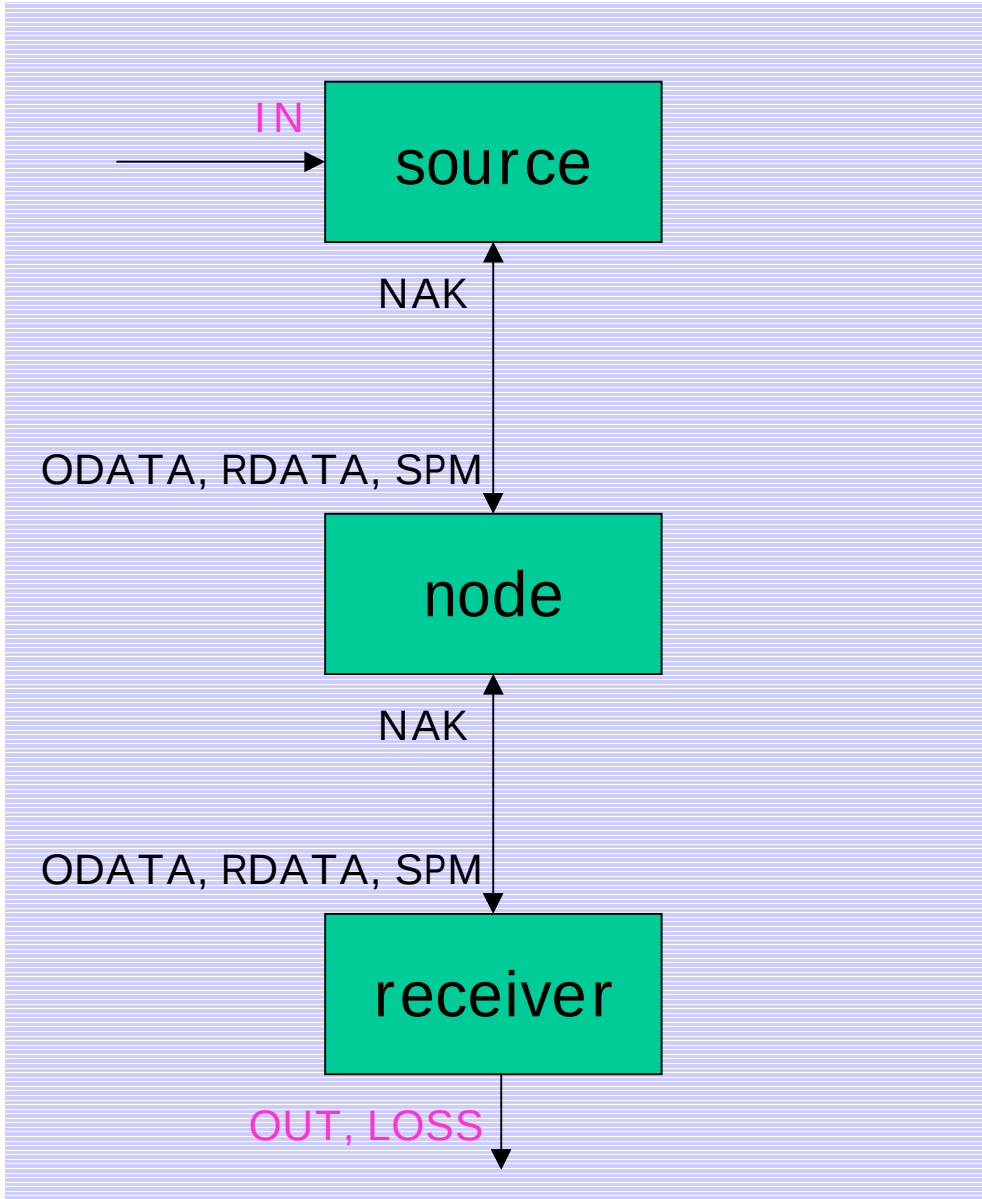
- source + node + receiver

- untimed models, few variables each
- several parameters

- only ODATA signals are lost
- NCF signal is not used

the models are written in IF-2.0

The protocol



The source

Parameters

MAX_RTE

MIN_RTE

Input queue

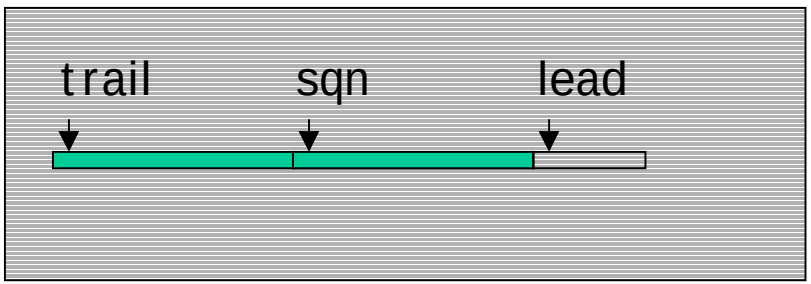
IN, NAK

Variables

int trail = 0 // trailing sqn no

int lead = 0 // leading sqn no

int sqn = 0 // next to be sent



?IN, lead-trail < MAX_RTE →

lead++

sqn < lead →

!ODATA(sqn, trail), sqn++

trail < sqn, lead - trail ≥ MIN_RTE →

trail++, !SPM(sqn-1, trail)

?NAK(x) →

if trail ≤ x and x < sqn then

!RDATA(x,trail) fi

The node

Input queue

ODATA, RDATA, SPM, NAK

Variables

set-of-int pending =∅

```
?ODATA(x,t) →
    !ODATA(x,t)

?ODATA(x,t) →
    τ

?RDATA(x,t) →
    if x ∈ pending then
        pending \= {x}, !RDATA(x,t) fi

?SPM(x,t) →
    !SPM(x,t), pending \= {y | y < t}

?NAK(x) →
    if x ∉ pending then
        pending ∪= {x}, !NAK(x) fi
```

The receiver

Input queue

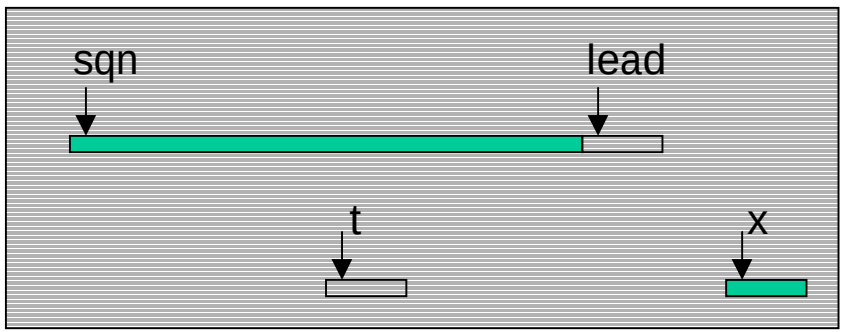
ODATA, RDATA, SPM

Variables

int sqn = 0 // next to deliver

int lead = 0 // next to receive

set-of-int window = ∅

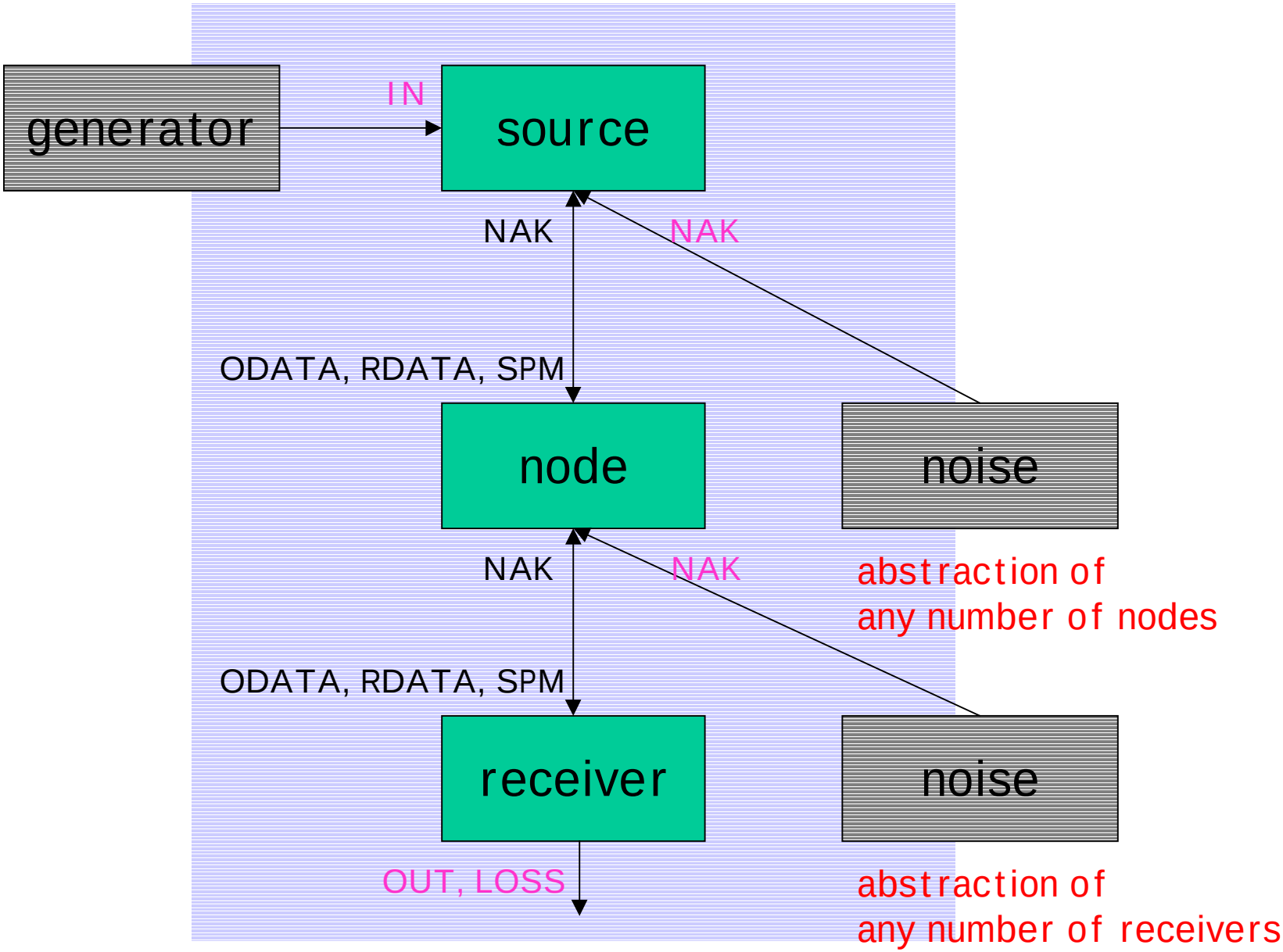



```
sqn ∈ window →
    window \= {sqn}, !OUT(sqn), sqn++

?XDATA(x,t) →
    foreach y ∈ [sqn, t)
        if y∈window
        then !OUT(y), window \= {y}
        else !LOSS(y) fi
    if sqn < t then sqn := t fi
    if lead < sqn then lead := sqn fi
    foreach y ∈ [lead, x)
        !NAK(y)
    if lead < x then lead := x fi

    window ∪= {x}
    if lead = x then lead++
```

The environment

- the data generator - send INs
- the noise - send NAKs
  - on source : abstracts any number of nodes
  - on node : abstracts any number of receivers

The PGM system



Fairness issue

The environment could be infinitely fast with respect to the protocol...

Solutions

- bounded queues - limit the input queues of protocol processes
- time - limit the number of messages that environment processes could send per unit of time
- priorities - give lower priorities to environment wrt to protocol processes

The generator

Parameters

MAX_SQN

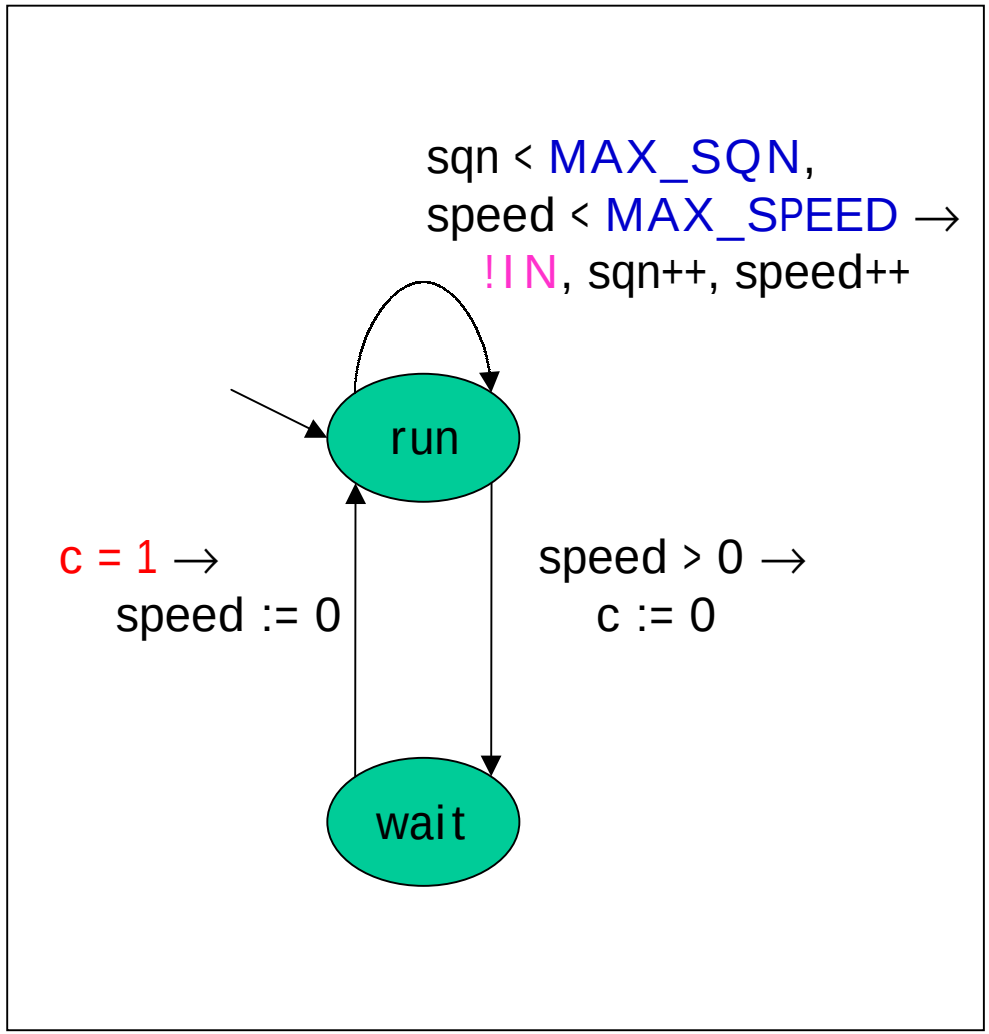
MAX_SPEED

Variables

int sqn = 0 // next to be sent

int speed = 0 // current speed

clock c = 0 //



The noise

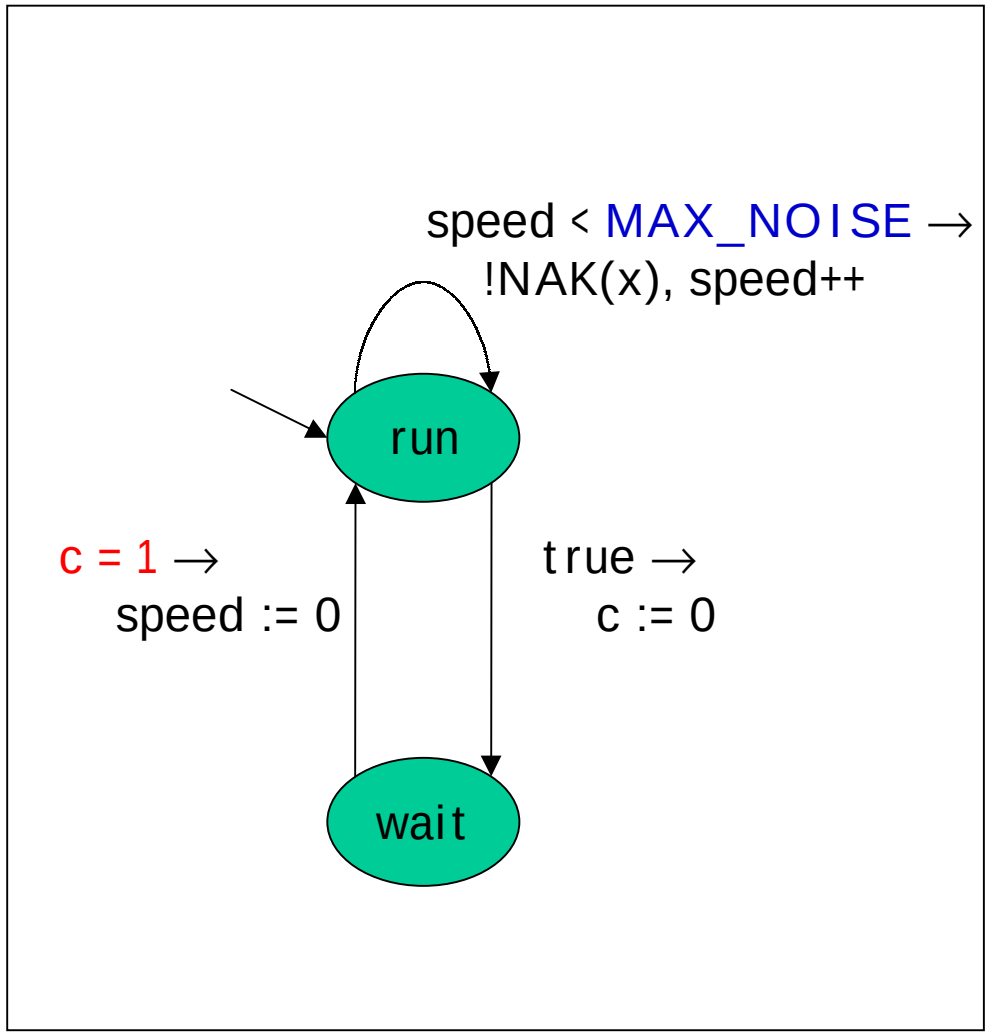
Parameters

MAX_NOISE

Variables

int speed = 0 // current speed

clock c



Verification

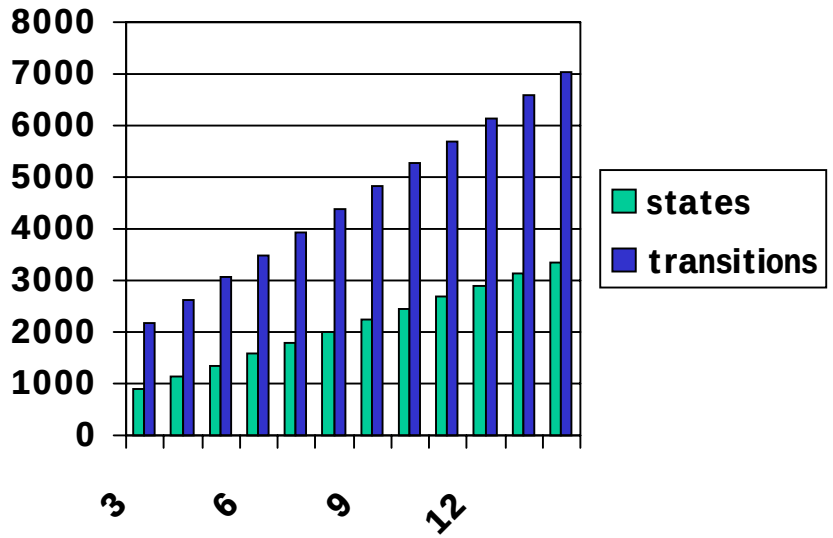
- state-space generation — IF-2.0 MC
  - without noise
  - with noise on source
  - with noise on node
- Model checking — Aldebaran
  - safety properties
  - model minimisation

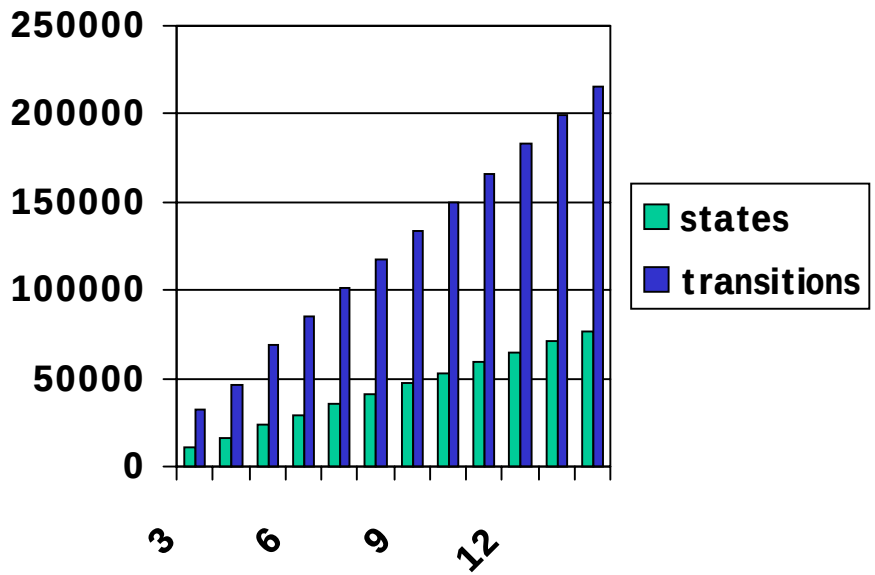
State-space generation

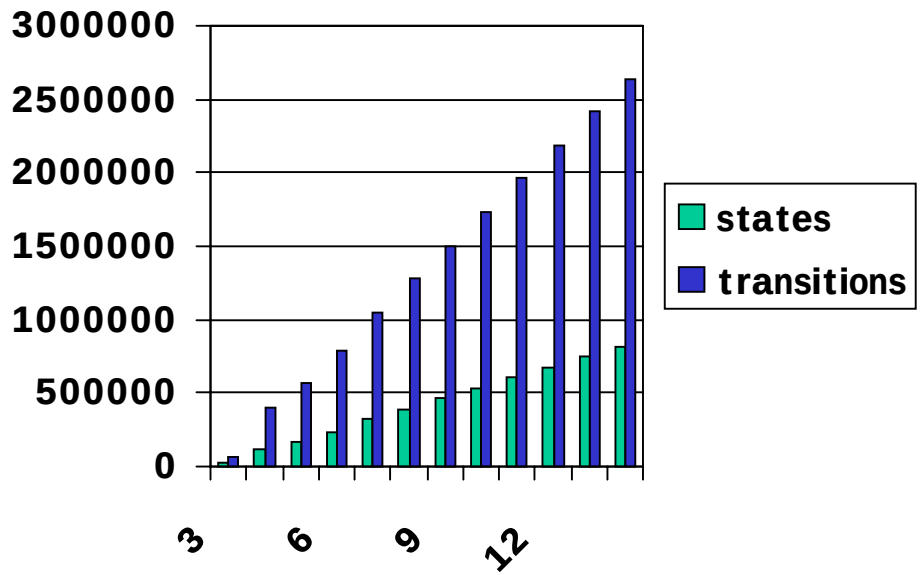
without noise

generator max speed =

{ 1, 2, 3 }





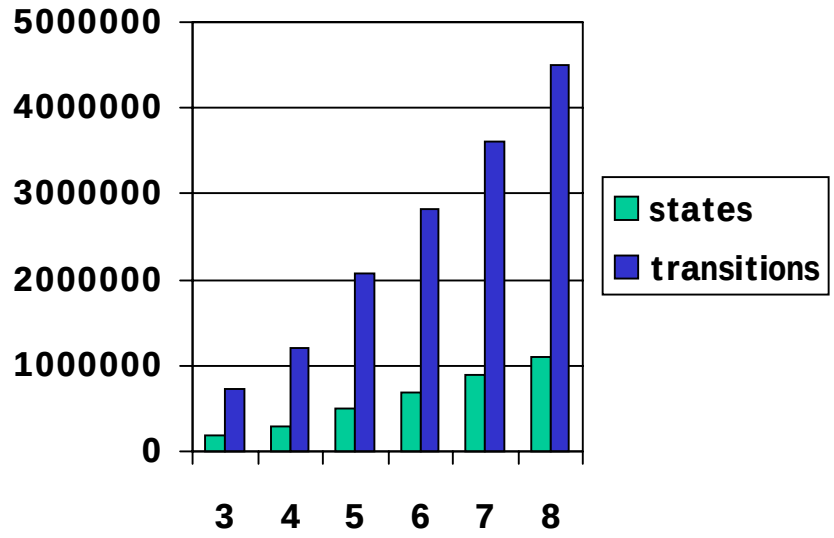


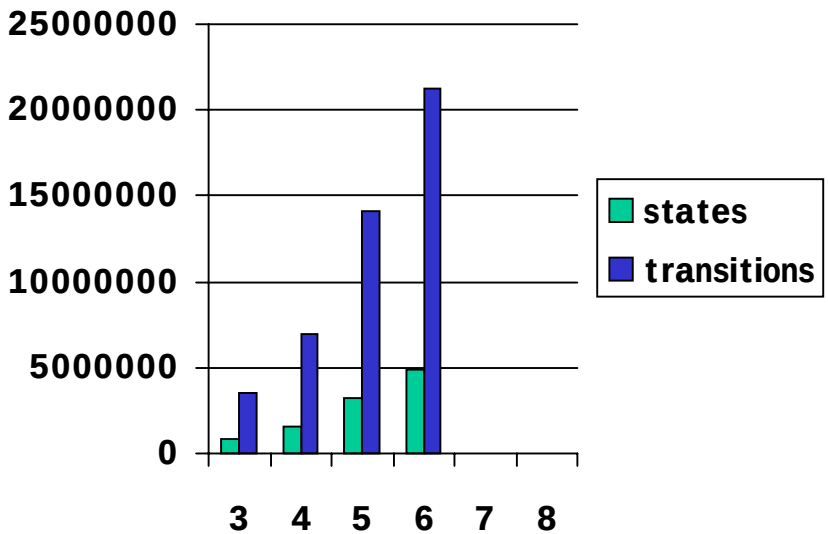
State-space generation

noise on source

generator max speed = 2

max noise = { 1, 2 }



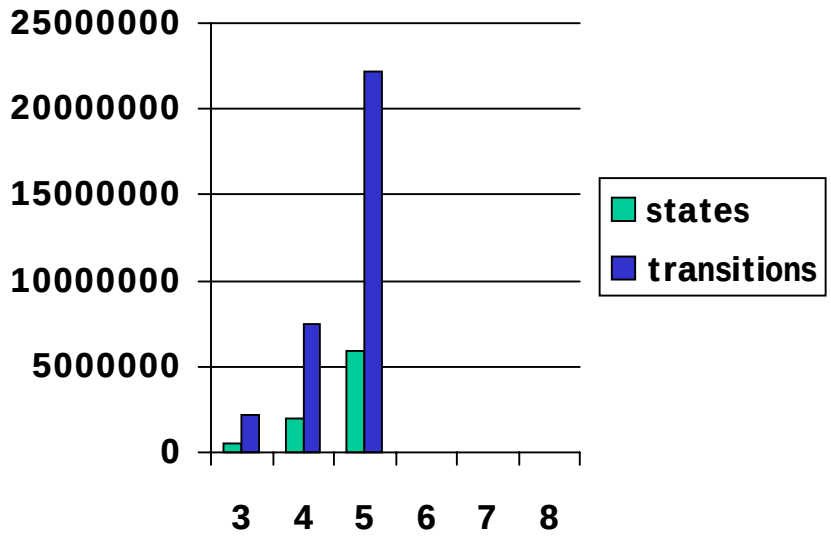


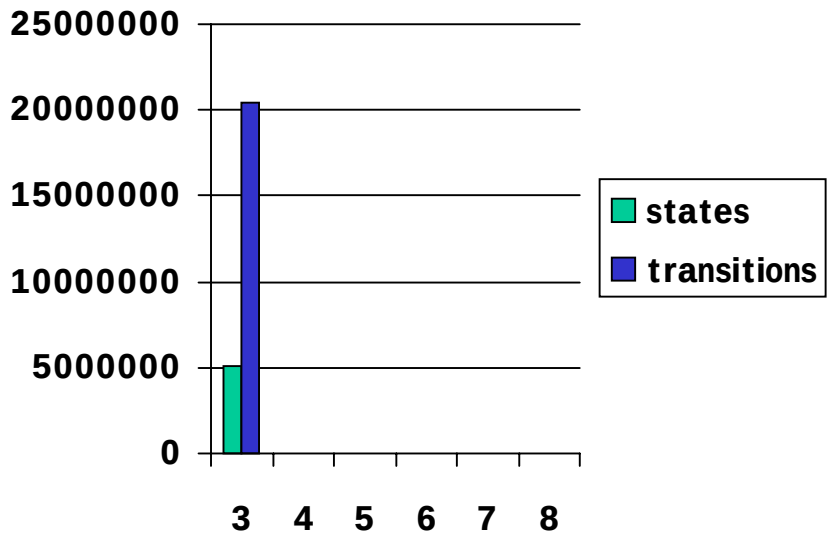
State-space generation

noise on node

generator max speed = 2

max noise = { 1, 2 }





Model checking

- message delivery

  $\forall i \in [0, MAX\_SQN)$

  eventually OUT(i) or LOSS(i)

- order preservation

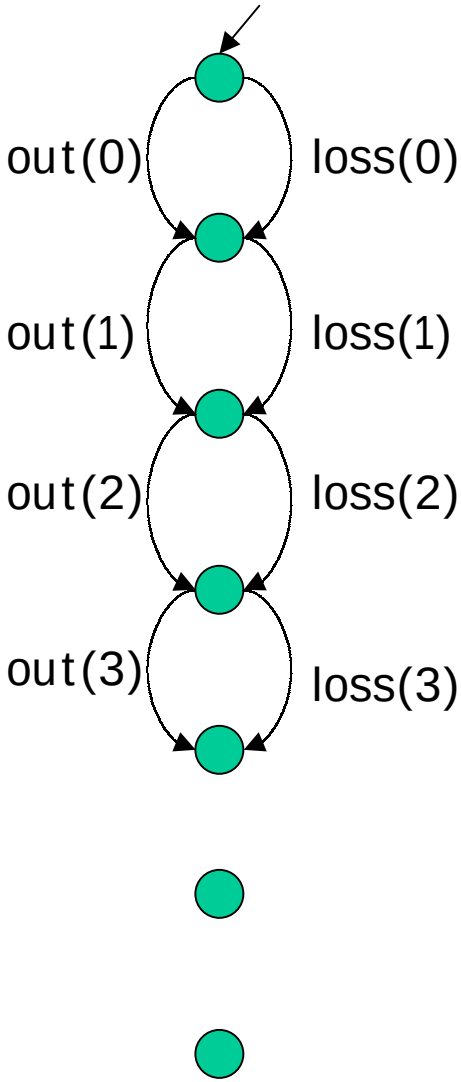
  $\forall i,j \in [0, MAX\_SQN)\ i < j$

  always OUT(i) or LOSS(i)

  before OUT(j) or LOSS(j)

Model checking

- no livelocks
- no deadlocks
- safety bisimulation reduction using **Aldebaran**
- **the two properties are verified on all generated models**



Conclusions

- we developed a simple model
  - abstract untimed model of PGM
  - reasonable timed model of the PGM environment
- verification by model-checking on fixed configurations
- model-checking is not enough

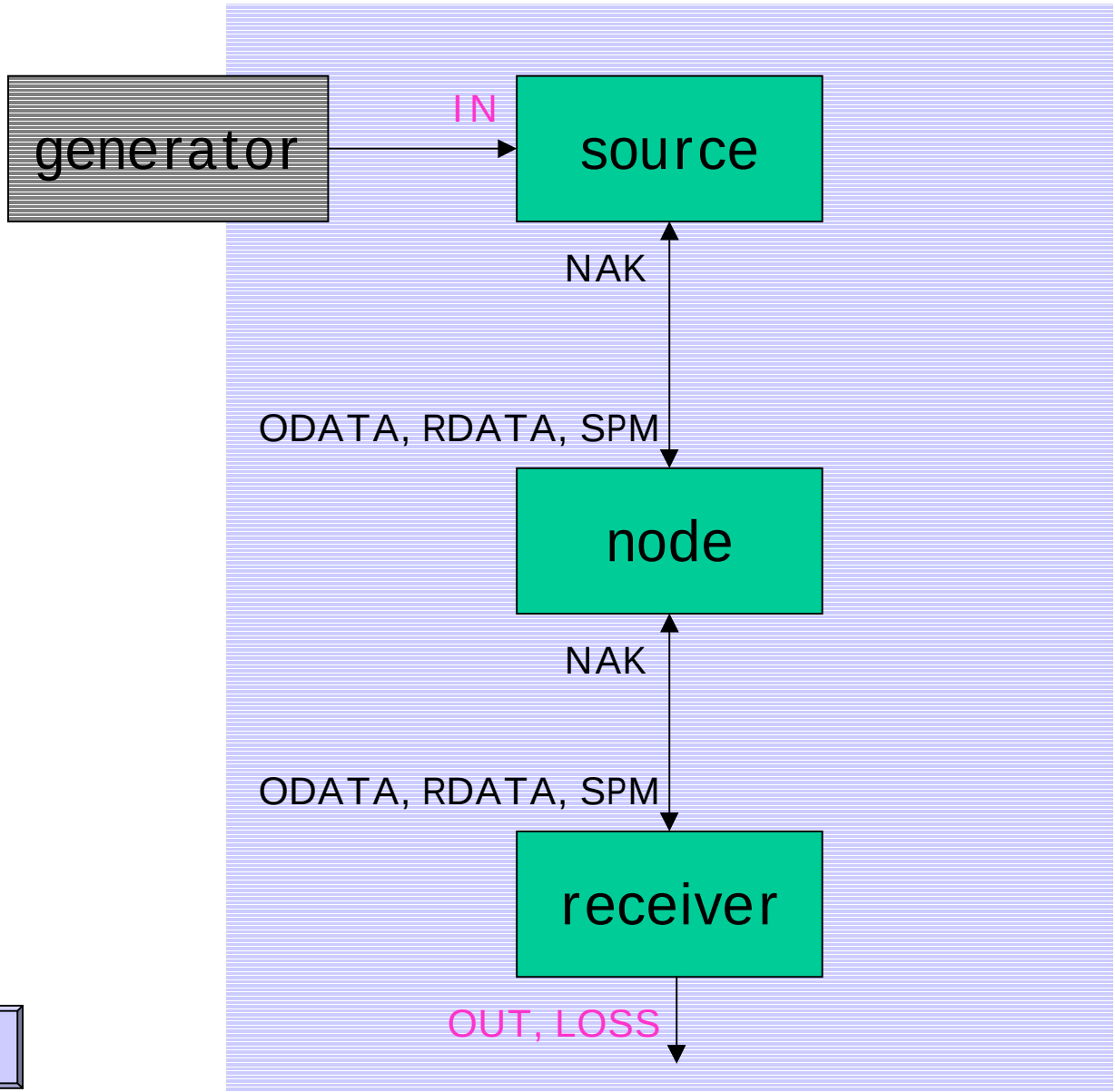
Future work

- extending the model i.e,
  - timing of the source
  - timing of communication

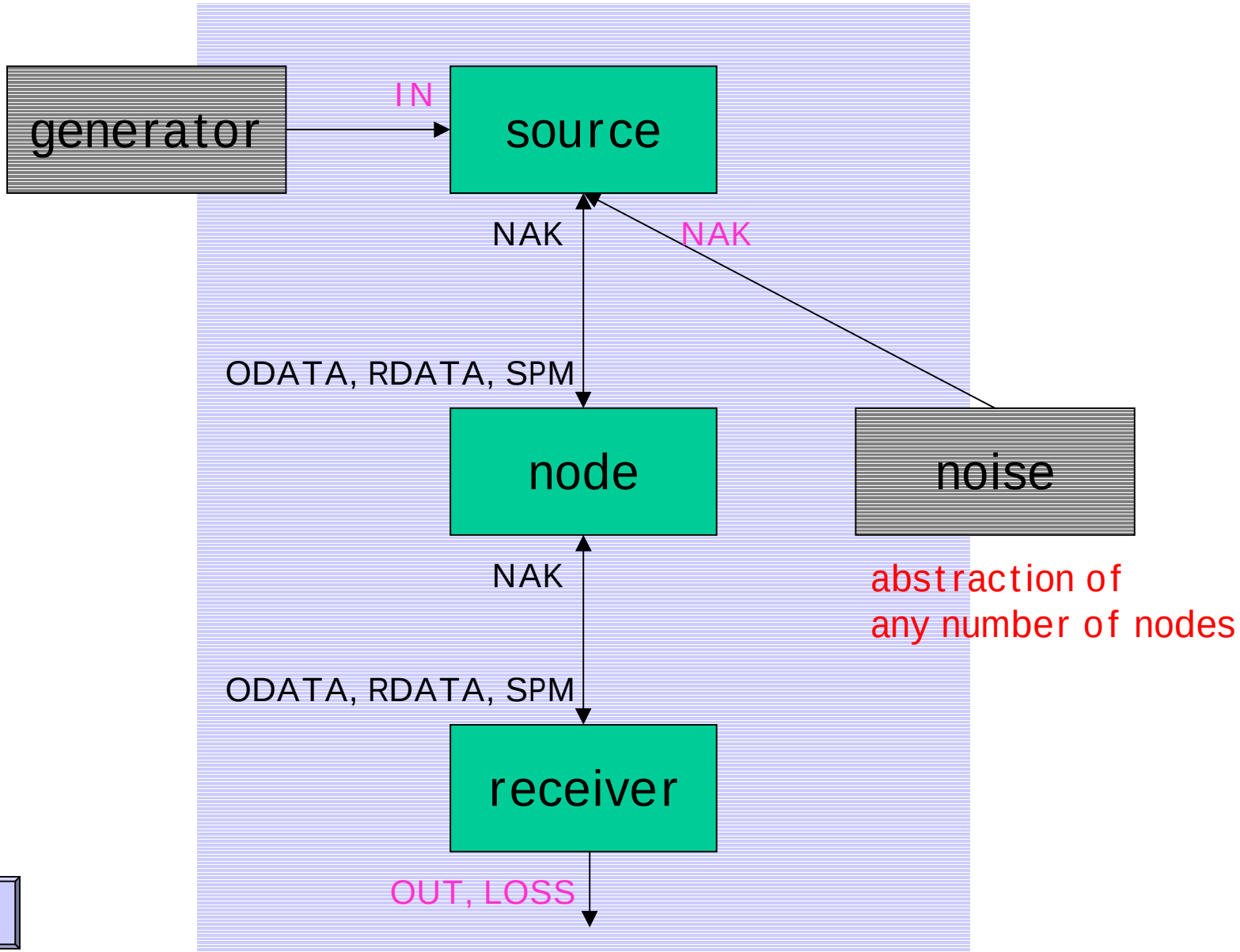
- try some symbolic analysis
  - abstraction
  - invariant computation
  - symbolic reachability

  the model is simple but heterogeneous

The PGM system



The PGM system



The PGM system