Evaluating Complex MAC Protocols for Sensor Networks with APMC

Authors:
Michaël Cadilhac Thomas Hérault Richard Lassaigne
Sylvain Peyronnet1, Sébastien Tixeuil

Presenter:
Surjya Ray
Outline

- Introduction
  - Probabilistic Model Checking
  - Approximate Probabilistic Model Checking (APMC)
- Approximate Probabilistic Verification with APMC
  - Notations Used
  - Approximate Verification
  - Generic Approximation Algorithm
  - APMC Implementation
- Sketch of the MAC protocol
- Modeling
- Experiments
- Results
- Conclusion
Probabilistic Model Checking

- A method to analyze the correctness and performance of MAC protocols.
- Features of this method:
  - It constructs a mathematical model of the system.
  - Expresses the required specifications in some temporal language.
  - Represents all possible system configuration including state transition probabilities.
- Problem: The transition matrix can become extremely large; verification becomes intractable giving rise to State Space Explosion.
Approximate Probabilistic Model Checking

Solutions include

- Symbolic and Numerical Methods
- Approximate Probabilistic Model Checking (APMC)\[^1\]

Using APMC, we can

- Compute the approximate probability that a model satisfies a specification.
- Significantly lower the memory consumption (or make it constant in some cases).
Approximate Probabilistic Verification with APMC

Features of APMC

• Uses sampling of execution paths of the system.
• Based on a randomized algorithm.
• Approximates the satisfaction probability of a temporal specification.
• Can approximate with any degree of accuracy, the satisfaction probability of a specification.
• Can handle any probabilistic system that can be modeled as a discrete–time Markov Chain
Notations Used

- $M$: system represented as a discrete-time Markov Chain
- $s$: initial state of the system
- $\psi$: linear temporal formula (specification or property) to be proved
- $Path(s)$: set of execution paths whose first state is state $s$.
- $Prob[\psi]$: Probability measure of the set of paths satisfying the formula $\psi$ in the set $Path(s)$. $Prob(.)$ is defined classically.
- $Path_k(s)$: Set of all paths of length $k > 0$ starting at $s$
- $Prob_k[\psi]$: Probability measure of the set of paths satisfying the formula $\psi$ in the set $Path_k(s)$.
Approximate Verification

In order to estimate probability $p = \text{Prob}_k[\psi]$ of a property $\psi$, we

- generate the random paths of $\text{Prob}_k(s)$.
- compute a random variable $X$ which estimates $p = \text{Prob}_k[\psi]$
- specify a real number $\varepsilon > 0$ which ensures that the estimation $X$ is $\varepsilon$-good, meaning output value of the algorithm lies in $[p-\varepsilon, p+\varepsilon]$.
- also specify another real number $\delta > 0$, which ensures that the approximation is $\varepsilon$-good with confidence $(1-\delta)$. 
Randomized Approximation Scheme (RAS) for $p$ is a randomized algorithm $A$ that takes as input a representation of the system $M$, a property $\psi$, two real numbers $\varepsilon, \delta > 0$ and produces a value $X$ such that

$$\Pr X \in p - \varepsilon, p + \varepsilon \geq 1 - \delta$$

Goal: Estimate $p = \text{Prob}_k[\psi]$
Generic Approximation Algorithm

Generic approximation algorithm $\mathcal{GAA}$

**Input:** diagram, $k$, $\psi$, $\epsilon$, $\delta$

**Output:** approximation of $\text{Prob}_k[\psi]$

$N := \ln\left(\frac{2}{\delta}\right)/2\epsilon^2$ ; $A := 0$

For $i = 1$ to $N$ do $A := A + \textbf{Random Path}(\text{diagram}, k, \psi)$

Return $Y = A/N$

where **Random Path** is

**Random Path**

**Input:** diagram, $k$, $\psi$

**Output:** samples a path $\pi$ of length $k$ and check formula $\psi$ on $\pi$

(i) Generate a random path $\pi$ of length $k$ (with the diagram)

(ii) If $\psi$ is true on $\pi$ then return 1 else 0
APMC Implementation

- Compiler
  
  *input*: model description $M$, $\psi$

  *output*: an ad-hoc verifier for the set of properties over the given model. These are in fact a set of ANSI C functions.

- Deployer
  
  *input*: $\delta$, $\varepsilon$ and $k$

  *output*: A stand alone binary. It creates the main function as well as the engine which the compiler output lacks
Sketch of the MAC Protocol

- TDMA like framed approach. Frames dived into slots.
- Two modes - **LooseMAC** and **TightMAC** [2]
- **LooseMAC**
  - Same frame size at all nodes
  - Simple
  - Lower throughput (due to large frames)
- **TightMAC**
  - Nodes may have different frame sizes
  - More complex
  - Higher throughput
LooseMAC- Basic Idea

- Nodes repeatedly select a **random time slot** until it is collision-free in the 2-neighborhood.
TightMAC- Basic Idea

- Nodes may have different frame sizes.
- Runs on top of LooseMAC.
- Motivation: “tighten” the frames to increase throughput.
Modeling

- Each sensor modeled by an independent module.
- Each sensor can be in one of the three modes:
  - NEWSLOT
  - WATCH
  - READY
- When all nodes are in READY state, system is stable
- The three main subparts of LooseMAC are
  - Send() : broadcasts state of node $i$ in current time slot.
  - Receive() : In every time slot, node checks for conflicts or fresh nodes.
  - UpdateMode() : updates mode of node $i$ in its current time slot.
Modeling

- The behavior of the main function of this TDMA protocol is as follows:

  **LooseMAC**
  
  (i) Initialize some internal values
  
  (ii) (a) Call sequentially Send(), Receive() and UpdateMode(),
       (b) Increment a local time reference,
       (c) Loop to (a).

- The events are implemented using the idea of *states*. A state describes the current progress of each module.
Modeling

- The equivalence between the execution of LooseMAC and the internal state of a node $i$ is as follows:

<table>
<thead>
<tr>
<th>Internal State</th>
<th>LooseMAC Main Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotIn</td>
<td>// Node is not in the network</td>
</tr>
<tr>
<td>Init</td>
<td>( \text{Mode} \leftarrow \text{NEW SLOT} )</td>
</tr>
<tr>
<td>Sending</td>
<td>( \text{Send()} () )</td>
</tr>
<tr>
<td>Receiving</td>
<td>( \text{Receive()} () )</td>
</tr>
<tr>
<td>UpdateMode</td>
<td>( \text{UpdateMode()} () )</td>
</tr>
<tr>
<td>Ended</td>
<td>( \text{TimeReference} \leftarrow \text{TimeReference} + 1 )</td>
</tr>
<tr>
<td></td>
<td>if ( \text{TimeReference} = \text{FrameSize} + 1 ) then</td>
</tr>
<tr>
<td></td>
<td>( \text{TimeReference} \leftarrow 1 )</td>
</tr>
<tr>
<td></td>
<td>end if</td>
</tr>
<tr>
<td></td>
<td>end while</td>
</tr>
</tbody>
</table>

- TightMAC is an addendum to LooseMAC code. It uses the slots found by LooseMAC to compute another conflict-free slot.
Experiments

- Parameters:
  - $\varepsilon = 10^{-2}$
  - $\delta = 10^{-5}$
  - $k = 32000$ (length of path)

- Topologies:
  - Peer-to-peer communication over dense graph
  - Peer-to-peer communication over sparse graph

- Experiments for LooseMAC
  - Contention free from initial state
  - A fresh node breaks the stability momentarily
  - Contention-freeness after a node joining the network

- Experiments for TightMAC
  - Stability of the network
Results: LooseMAC

(a) prob. vs Time Units - dense graph - frame size = 32

(b) prob. vs Time Units - sparse graph - frame size = 32
Results: LooseMAC

(a) prob. vs Time Units - dense graph - frame size = 64
(b) prob. vs Time Units - sparse graph - frame size = 64
Results: TightMAC

Fig. 5. Experimental results - TightMac
Conclusion

This paper presents an analysis, using approximate probabilistic model checking, of a TDMA based contention-free MAC protocol.

- This method allows to efficiently verify/analyze the correctness and performance of complex distributed algorithms over sensor networks.
- This method does not suffer from the state space explosion phenomenon arising with classical model checking methods.
- However the numerical results are accurate only with respect to an approximation parameter ($\varepsilon$ here).
- The method saves a lot of computing power and memory.
References

Thank You.
Any Questions?