Congestion Verification on Abstracted Wireless Sensor Networks with the WSN-PN Tool

Khanh Le, Thang Bui, Tho Quan, Laure Petrucci, and E ́tienne Andre ´

***Abstract*—The paper presents the WSN-PN tool, which aims at modelling and verifying Wireless Sensor Networks (WSN) using Petri nets (PN). Especially, WSN-PN allows for congestion detection on a WSN setting. Moreover, WSN-PN supports users to abstract components, which can be either sensors or channels, on the verified PN. This abstraction is possible due to the observation that in a practical situation, a reason that causes a WSN to be congested is only depending on either sensors or channels. As a result, once abstracted properly, the verification speed is improved significantly, as illustrated in our experiments.**

***Index Terms*—WSN, WSN-PN tool, Petri nets, congestion detection.**

I. Introduction

A *Wireless Sensor Network* (WSN) is a collection of hundreds or thousands of *Sensor Nodes* (SNs), or *sensors*. A SN component consists of sensing, computing and communicating elements. They are connected to each other by a wireless interface. Nowadays, SNs can be considered as cheap, low energy, limited memory and capacity of processing [1]. Battery is the primary power resource of SNs. Some sensors also have a secondary power resource which is harvested from the light.

WSNs are deployed according to a *dense* or a *sparse* mode to cover a lot of application systems. Environmental systems used to monitor the weather, the temperature, the pressure and habitat, systems such as animals monitoring and tracking are usually implemented using a dense network topology [2]. Some applications need sensors spread over a large geographical area in a sparse deployment such as a sensing system at a city intersection for tracking transportation or habitat monitoring [3].

In the wireless environment, the network which is established by wireless connection is unstable than that of wired networks. Due to the unstable connection, packets are transmitted for several times and may cause network congestion. Moreover, applications such as multiple-objects tracking which are usually deployed with dense topology, generate countless data transmissions, and thus may suffer from this problem [2].

Manuscript received September 6, 2015; revised January 22, 2016.

Khanh Le, Thang Bui, and Tho Quan are with University of Technology, Ho Chi Minh, Vietnam (e-mail: lnkkhanh@@cse.hcmut.edu.vn, thang@cse.hcmut.edu.vn, qttho@cse.hcmut.edu.vn).

Laure Petrucci and E ́tienne Andre ´ are with Universite ´ Paris 13, Sorbonne Paris Cite ́, LIPN, CNRS Villetaneuse, France (e-mail: laure.petrucci@lipn.univ-paris13.fr, etienne.andre@lipn.univ-paris13.fr).

Generally, congestion can be easily detected in dense deployments. However, it still occurs in sparse mode [4], [5]. In dense mode, congestion occurs due to the overload of buffer size in sensor nodes and the collision of packets over the transmission medium. However, congestion occurs more frequently at the channel in sparse mode due to interference [4]. Based on the scenarios given from experimentations in CODA [4], the causes of congestion in WSNs can be: *i*) buffer overload on sensors in dense WSNs; *ii*) packet collision on channels in dense WSNs; and *iii*) interference on communication channels in sparse WSNs. Thus, to detect congestion in a concrete situation, one only needs to examine information on a specific type of component, either sensors or channels, instead of the whole WSN, which should be costly due to its complexity.

The paper makes this idea feasible by introducing a tool named WSN-PN. This tool allows users to model a WSN (using a domain specific input for WSNs), which will then be translated into a Petri net (PN) [6]; then WSN-PN verifies congestion on the PN model by means of model checking. In WSN-PN, users do not need to work with the details of the PN model. Instead, they only need to specify the topology and parameter setting of a WSN; the corresponding PN will then be generated automatically. Also, WSN-PN supports *component abstraction* of a WSN. That is, users may choose to abstract sensors or channels in the PN model and verify the remaining part. This approach thus significantly reduces the verification complexity when performing congestion checking. Our experiments show that several WSN models for which traditional verification approaches suffered timeout have been successfully verified for congestion when abstracted properly using WSN-PN.

It is notable that, even though WSN-PN relies on Petri net (a general and popular modelling language) to perform model checking, this tool is specifically intended for WSNs, for the following reasons.

- WSN-PN supports specifying parameters of a WSN, as illustrated in the following sections. Based on the parameters given, a corresponding Petri net will be automatically produced.
- WSN-PN supports abstracting certain items of a WSN, such as sensors and channels. Note that users can choose to abstract WSN items, not a sub-Petri net on the generated model. That is, this tool does not require users to have advanced knowledge of Petri nets.

Outline: The rest of the paper is organized as follows. Section II discusses related works. Section III introduces our tool WSN-PN, and explains how to model a WSN and to generate PN model. Section IV shows in details how the tool detects congestion. More extensive experiments are reported in Section V. Finally, Section VI draws conclusions and outlines future work.

II. Related Works

A. Congestion Detection Algorithms

Most congestion detection algorithms use a buffer/queue as main key for computation. Siphon [7] is a congestion mitigation scheme which detects congestion by using queue length. But instead of using any rate adjustment technique, it uses traffic redirection to mitigate congestion.

Congestion Detection and Avoidance (CODA) [4], uses both buffer threshold and buffer weight for detection, and combines them with a bottleneck-node-based method to control the sending/receiving packets. WSN-PN adopts this approach.

In Fusion [5], congestion is detected in each sensor node based on a measurement of the queue length. The node that detects congestion sets a congestion notification bit in the header of each outgoing packet. Once the congestion notification bit is set, neighbouring nodes can overhear it and stop forwarding packets to the congested node so that it can drain the backlogged packets.

B. Congestion Detection Tools

To understand congestion detection activity, most algorithms were simulated on a simulator. A simulator is a tool used to simulate performance or validate some properties of networks such as delay, packet loss, congestion and so on. The current simulators widely used include ns2[1] or OMNeT++ [8]. In another aspect, Simulink[2] is a commercial software that generally allows user to model a system from basic blocks and write code in various programming languages to simulate the model operations. The tool also supports analysis of the simulation results applicable for WSN modelling.

In these simulators, a WSN is modelled by its sensors and channels first, after that users can analyze the properties such as QoS constraints or simulate the action of protocols. All activities are done by the supporting of an appropriate *framework*. For example, in OMNeT++, the *mf* framework was used first, and then changed to the *inet* framework. In *mf* framework, the routing protocol is specified in Network layer, where as in *inet*, this is described in Application and Transport layers. Obviously, the changing of supported framework affects the using of framework significantly as users must completely change all their predefined models even though the WSN topology and parameters remain the same. Moreover, network programmers need time to adapt and learn how to use the framework to write their experiments.

In our approach, we use a Petri net to model a WSN and perform formal verification to detect possible congestion. This approach can overcome the problem of simulation, but theoretically suffers from a huge computational cost. We try to handle this by allowing users to abstract elements of a WSN, *i.e.* sensors or channels, when they are not the cause of congestion. The immediate advantages of such an approach are twofold:

1) It alleviates the dependence on the simulator framework since the WSN is modelled at a higher level of abstraction, which only includes sensors and channels. Thus, the WSN model is always the same, independent of the framework being used.
2) It defines all scenarios and verifies the properties by model checking a logic formula while simulators must be done by programming.

As a result, our tool can verify some real WSN settings, which the conventional formal approach fails to handle due to the state space explosion.

III. Verifying WSNs with WSN-PN

This section presents the WSN-PN tool to analyze the congestion of a WSN through its PN model.

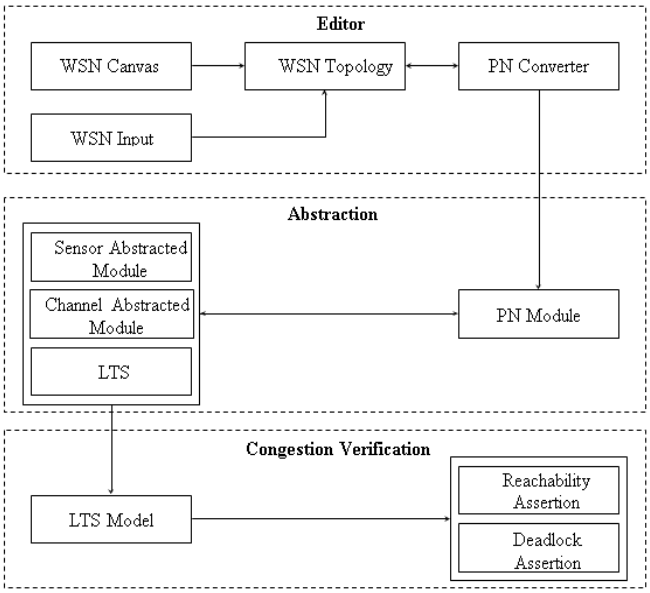


Fig. 1. WSN-PN architecture

A. Architecture of WSN-PN

Fig. 1 gives the architecture of WSN-PN, which consists of the following modules:

1) Editor

It helps users to describe a WSN by its topology as well as to set the initial parameters for the specified network. Then, the corresponding PN is generated automatically.

2) Abstraction

This module abstracts the original PN into an *abstracted PN*, as explained in the next section. Users can choose to produce *sensor-abstracted model* or *channel-abstracted model* using this module.

3) Congestion Verification

This module is in charge of verifying whether congestion occurs in a concrete or abstracted PN. It relies on the PAT model checking library [9]. The congestion condition is specified as an LTL formula, and the PAT model checking library is then employed to perform the verification.

B. WSN Modeling and Parameter Setting

As discussed, a WSN consists of several *sensors* that can communicate with each other using Wi-Fi signals. There are three types of sensors: *source*, *sink* and *intermediate node*. The role of intermediate nodes is to receive and

[1]http://www.isi.edu/nsnam/ns/
[2]https://fr.mathworks.com/products/simulink/index.html?s tid=gn locdrop/

forward packets, as depicted in the oil monitoring application reported by [10]. In some applications, the role of source and intermediate sensors are the same, *i.e.* they both can generate and send packets. In that case, it could be modeled as a combination of a source node and an intermediate node in WSN-PN. These sensors can be connected in *unicast*, *multicast* or *broadcast* mode, each of which specifies whether certain pairs of sensors can exchange information or not. If two sensors can communicate, we say that there is a *channel* established for these sensors. Information on sensors and channels forms the topology of a WSN. An example of a network topology is given in Fig. 2, illustrating a WSN consisting of 10 sensors. These sensors play the roles of intermediate nodes, conveying information from a source (denoted as double-lined circle) to a sink (denoted as a full circle).

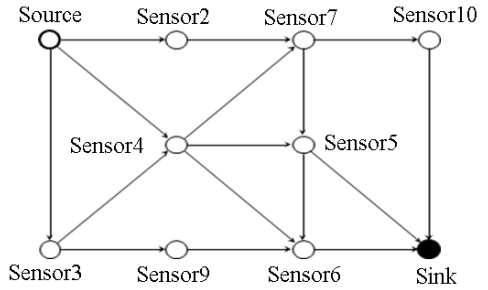


Fig. 2. A WSN with 10 nodes in unicast mode.

C. Component Encoding and Operational Semantics

In our approach, each physical sensor is encoded as a tuple of {*B*, *Q*, *p*, *s*}, where *B* is a buffer storing incoming packets, *Q* is a queue keeping processed packets ready to be sent out, *p* is *processing rate* specifying the rate of packets being transferred from *B* to *Q*, whereas *s* is the *sending rate* of packets being sent from *Q* to the channels connected to the sensor. We adapt the idea of buffer and queue to [11] The processing rate indicates the number of packets a sensor can handle (transfer from buffer to queue) over a given period of time, while the sending rate specifies the number of packets sent by a sensor to its connected channels. The idea of processing rate and sending rate are extracted from MICA, famous sensor node architecture to achieve high communication bandwidth with the flexibility to efficiently implement novel communication protocols [12]. These parameters are configurable in WSN-PN.

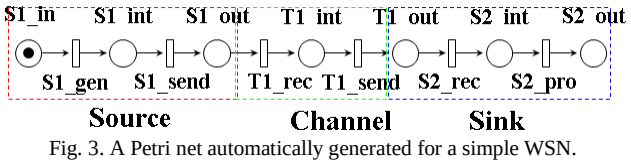


Fig. 3. A Petri net automatically generated for a simple WSN.

Meanwhile, each channel is also encoded as a pair {B_c, t}, where B_c the buffer storing packets being processed in the channel, and *t* is the *transmission rate* of packets that the channel can manage to process (*i.e.* sending out to connected sensors). The transmission rate of a channel connecting two sensors can be estimated based on the sensors estimation, as introduced in [13]. Subsequently, the sending rates are randomized as a range suggested by the empirical study in [4]. Table I gives an example of parameters set for *Source* and *Sink* sensors in Fig. 3.

TABLE I: EXAMPLE OF PARAMETERS SETTING FOR A SIMPLE WSN

Node	Source	Sink
Sending rate	2-3 packets/ms	N/A
Buffer size	50 packets	100 packets
Packet size	1KB	1KB
Processing rate	2-3 ms/packet	1-2 ms/packet

The operational semantic rules of the encoded sensors and networks are presented in Table II.

As discussed, each sensor and channel is modeled a component Petri Net, which is further abstracted as an abstract PN if needed. Thus, the same encoding mechanism and operational semantic rules are applied for the PN model of a WSN and its abstracted counterparts. This allows us to verify the abstracted models for congestion, instead of the original models.

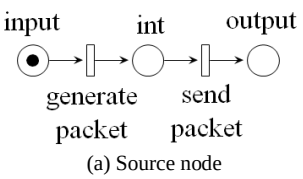
TABLE II: OPERATIONAL SEMANTIC RULES OF THE MODELLED WSN

Rules	**Explanation**
$\frac{\{B,Q,p,s\},\{B_C,t\},Q>0}{Q=Q-\frac{1}{s},B_C=B_C+\frac{1}{s}}$ [sensor-to-channel]	This rule is applied when a sensor sends packet to connected channel
$\frac{\{B_C,t\},\{B,Q,p,s\},B_C>0}{B_C=\max\ (0,B_C-\frac{1}{t}),B=B+\frac{1}{t}}$ [channel-to-sensor]	This rule is applied when a packet is transmitted to a sensor via a channel
$\frac{\{B,Q,p,s\},B>0}{B=\max\ \left(0,B-\frac{1}{p}\right),Q=Q+\frac{1}{p}}$ [sensor-proccessing]	This rule is applied when a packet is internally processed within a sensor

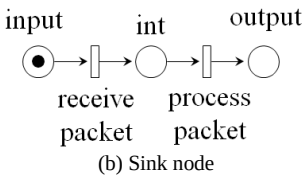
D. Petri Net Generation

WSN-PN supports generating a Petri net from a WSN, whose topology is specified by the user. Sensors and channels are first modelled individually as *component Petri nets*. Fig. 4 depicts the component Petri Nets.

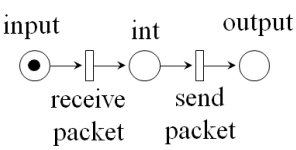
To model a whole WSN, WSN-PN automatically generates component Petri nets for each sensor and channel described in the topology and combines them together to obtain the corresponding PN of the complete WSN. For example, Fig. 4 presents the PN automatically generated for a simple WSN consisting of one *Source* and one *Sink*, which are connected via a channel *C*. The sensors and channel are represented by their corresponding component PNs. When the numbers of sensors and channels increase, the resultant PN also does, as in Fig. 5. It urges us to propose the abstraction approach as subsequently discussed.



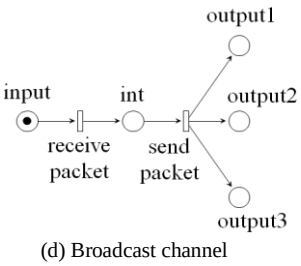
(a) Source node



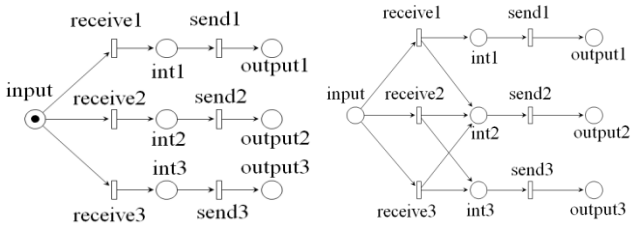
(b) Sink node



(c) Intermediate



(d) Broadcast channel



(e) Unicast channel (f) Multicast channel

Fig. 4. Corresponding component Petri net models of sensors and channels.

E. Abstraction

As discussed in Section I, in many cases we do not need to consider a full WSN for congestion detection, but only either sensors or channels. WSN-PN supports the abstraction of the non-necessary components in a WSN for a more efficient verification. For example, in Fig. 6, *Source* and *Sink* are abstracted as individual places, depicted larger and dashed, in case we only need to consider channels of the WSN for verification. Similarly, in Fig. 7, *Channel* is abstracted as an abstracted transition. For the full example modelled by the Petri net of Fig. 5, Fig. 8 and Fig. 9 illustrate the cases where the sensors and the channels are abstracted, respectively.

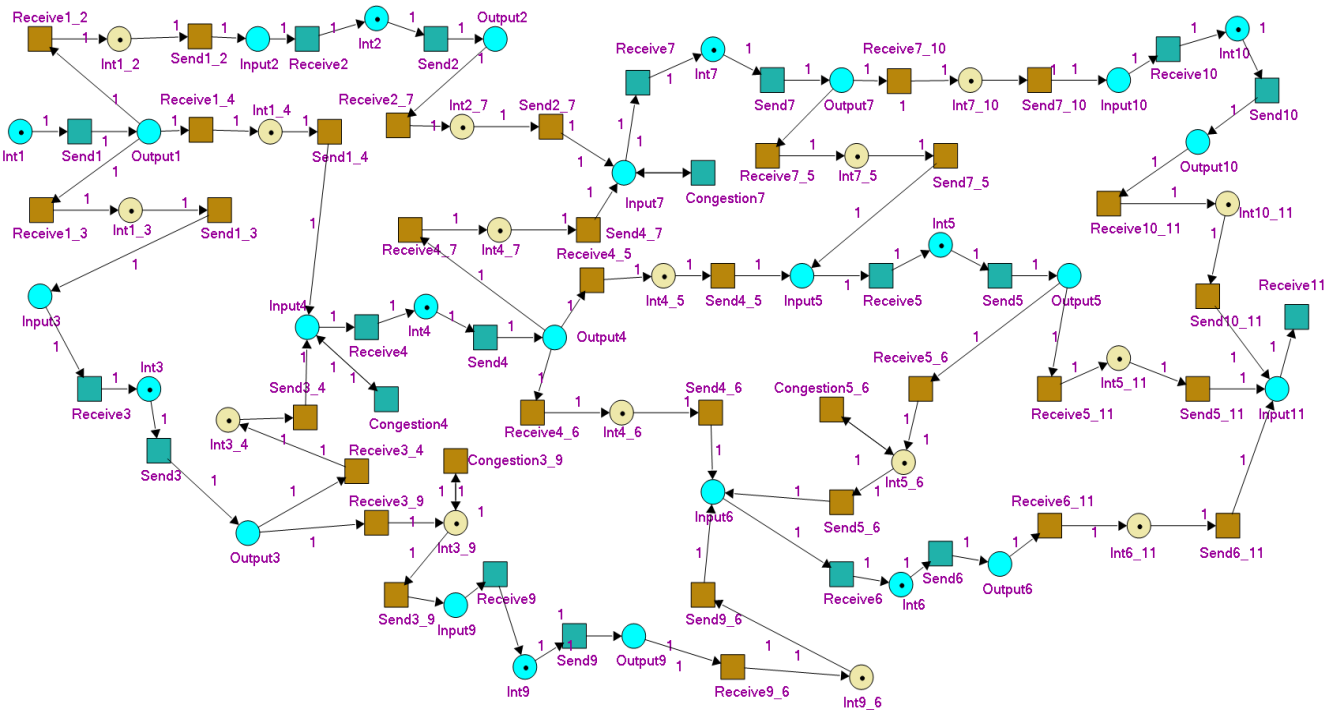


Fig. 5. The PN generated from the WSN in Fig. 3.

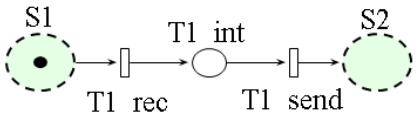


Fig. 6. The PN in Fig. 4 sensor-abstracted.

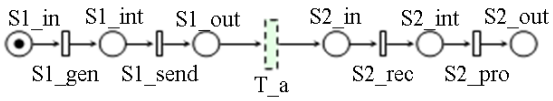


Fig. 7. The PN in Fig. 4 channel-abstracted.

Note that our abstraction method is an under-approximation, in the sense that a congestion case detected in the abstracted model always corresponds to a congestion occurring in the original model (*i.e.* no false positive occurring). This is because the abstracted WSN elements do not affect the probability of congestion in the verified situation, as stated by studies of WSN presented in Section I.

F. Component-Based Concurrent Processing

WSN-PN uses model-checking approach to verify congestion on a PN-modeled WSN. This technique verifies whether a property holds on a model by exploring all of possible *states* of the model to check whether the property holds at any state or not.

Typically, the traditional way to model-check a PN model is to explore all markings of the model, each of which is treated as a state. However, in the case of WSN verification, WSN-PN needs to ensure that the WSN model works properly in terms of *timing*. To better illustrate this, let us consider the following running example depicted in Fig. 10, which is a simple WSN whose corresponding sensor-abstracted PN model is given in Fig. 11. The marking given in Fig. 12(a) presents a situation when Sensor 1 sends packets to Sensor 2 and Sensor 3 in broadcast mode. From here, there are several possible markings can be generated. In Fig. 12(b) is the marking presenting the situation that Sensor sends packets to Sensor 4, and then Sensor 4 further forwards the packets to Sensor 5 as illustrated in Fig. 12(c).

However, in the real situation, as Sensor 2 and Sensor 3 received packets from Sensor 1 and then continue sending those packets to Sensor 4 at almost the same time, Sensor 4 should only send packets to Sensor 5 after receiving packets from both Sensor 2 and Sensor 3. In other words, the marking introduced in Fig. 12(c) is not feasible and should not be verified.

As a PN model in WSN-PN is composed from components, we deal with this situation by enforcing the

concurrent mechanism as follows. At a certain state corresponding to a marking, a new state is introduced by *firing all of currently-enabled transitions in all components*. This simulates the real operational mechanism that all components are working concurrently in the real situation. Thus, at the marking presented in Fig. 12(a), as there are two enabled transitions in two channels (note that channels are the only remained components on the model after the sensors are abstracted) connecting Sensor 2 and Sensor 3 to Sensor 4, both transitions are then needed to be fired to introduce a new state corresponding the marking illustrated in Fig. 12(e). The order for firing these transitions thus does not matter. After that, Sensor 4 continues sending packets to Sensor 5, introducing a new state as depicted in Fig. 12(f).

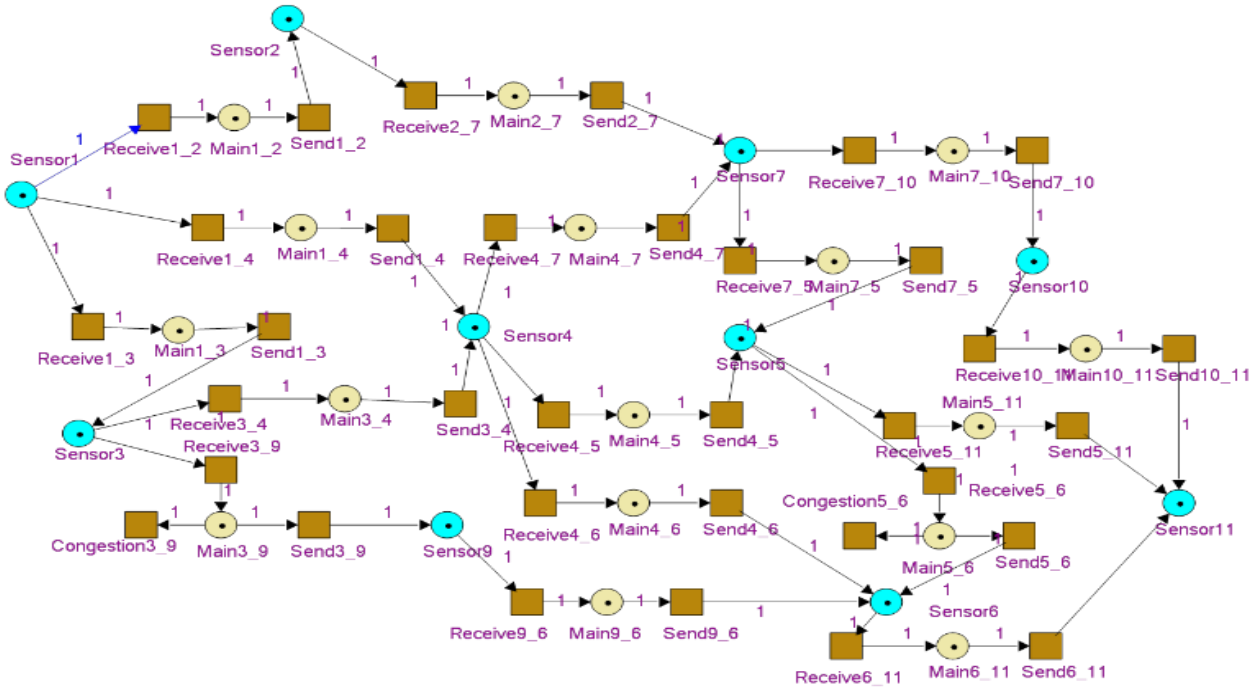


Fig. 8. The PN in Fig. 5 sensor-abstracted.

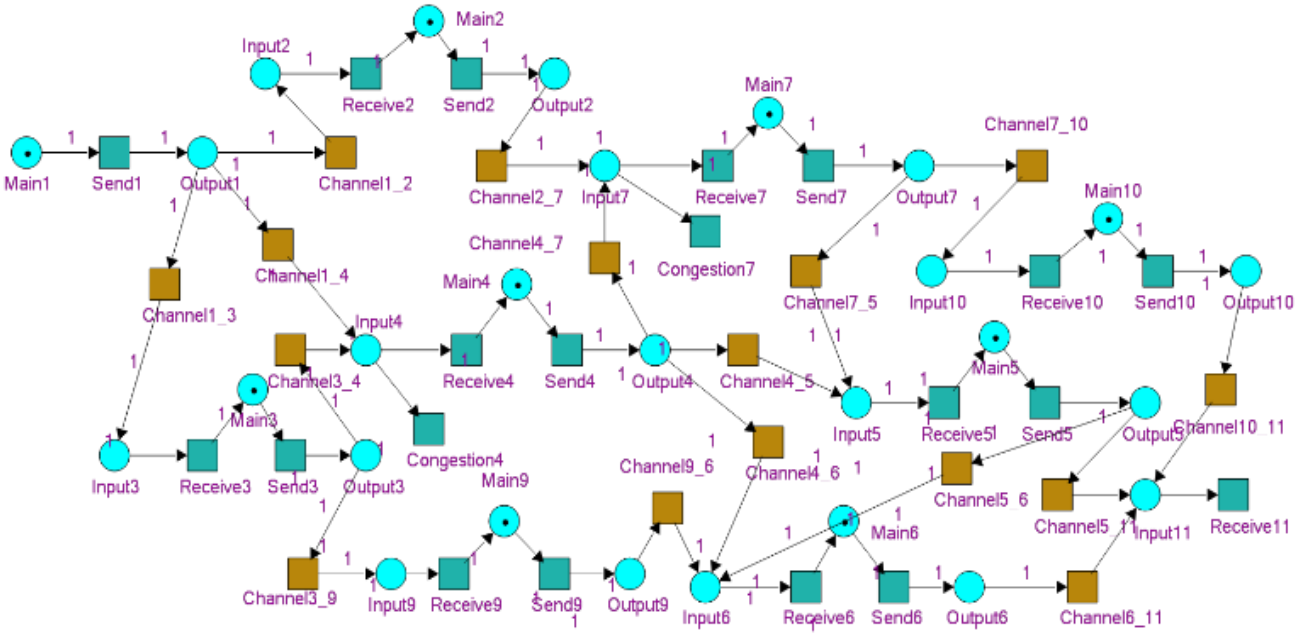


Fig. 9. The PN in Fig. 5 channel-abstracted.

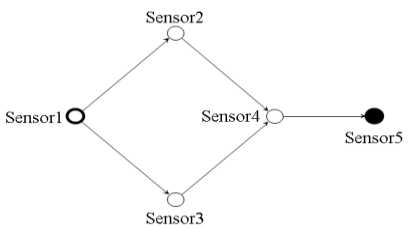


Fig. 10. Another WSN example.

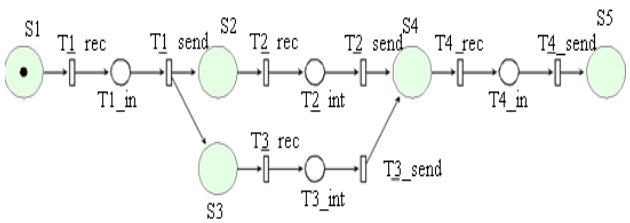
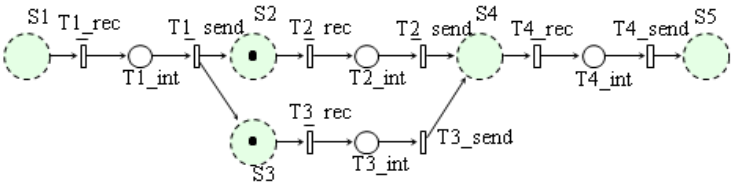
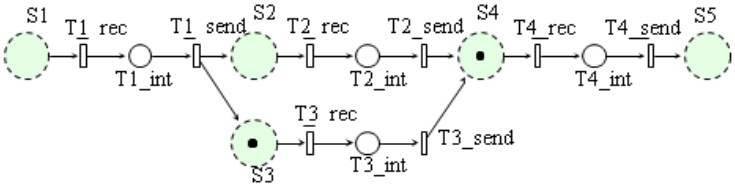


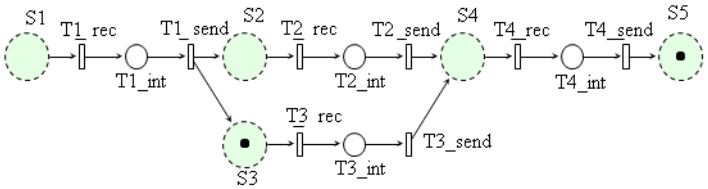
Fig. 11. PN generation in broadcast mode (sensor-abstracted) of Fig. 10.



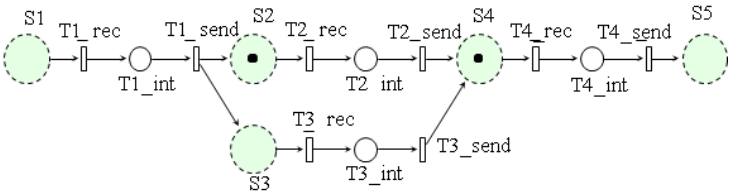
(a) Marking when Sensor 1 sends packets to Sensor 2 and Sensor 3 simultaneously (feasible marking).



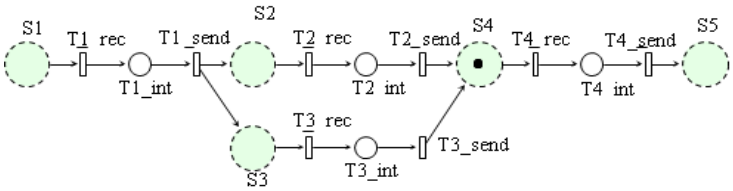
(b) Marking when Sensor 2 sends packets to Sensor 4 (feasible marking but not introducing new state).



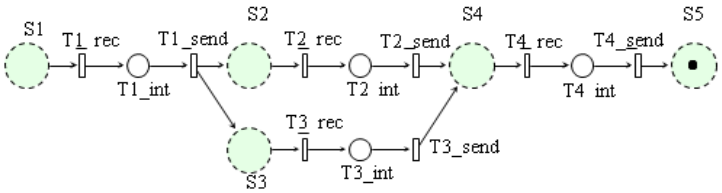
(c) Marking when Sensor 4 sends packets to Sensor 5 before receiving packets from Sensor 3 (infeasible marking since such a situation should not occur in a real WSN).



(d) Marking when Sensor 3 sends packets to Sensor 4 (feasible marking but not introducing new state).



(e) Marking when Sensor 4 received packets from both Sensor 2 and Sensor 3 (feasible marking introducing new state since all of enabled transitions in each component have been fired).



(f) Marking when Sensor 4 sends packets to Sensor 5 after receiving packets from both Sensor 2 and Sensor 3 (feasible marking introducing new state).

Fig. 12. Markings of Fig. 10 in broadcast mode (sensor abstraction).

IV. Congestion Detection

Basically, WSN-PN can check any property on a WSN, as long as the property can be expressed as an LTL formula. Thus, to check the congestion, one can develop an LTL formula as follows.

#assert WSN() |= []<> Congestion

where []<> Congestion stands for the LTL operations of □◊ (which means *always eventually*) and the condition

Congestion implied a property of whether a congestion occurs or not. The valuation of whether *Congestion* holds or not at a certain checked state is simulated by C# code as follows. To detect congestion on a sensor, our simulated code counts the number of received packets at the sensor. If this number reaches a threshold (*i.e.* greater than 70% buffer size, based on CODA conclusions), the guard condition lets the flow reach a special state making *Congestion* hold. A similar method is applied for detecting congestion in channels. *Simulated code to check buffer overload* The C# source code to check whether a sensor's buffer is full (*i.e.* causing congestion) is as follows.

```
public bool isFullSensor (int id){
   return ( sensors[id].PBuffer.Count >=
   sensors[id].BufferMaxSize);
}
```

Using this method, WSN-PN can detect whether congestion occurs in the WSN using the simulation code as previously discussed. The WSN with the parameters given in Table I is congestion-free. However, if one modifies the *Buffer* size parameter of *Source* from 100 to 150 packets, congestion will occur, due to buffer overload. This congestion is detected by WSN-PN. Moreover, instead of verifying the full WSN in Fig. 4, this congestion can also be detected on the channel-abstracted version given in Fig. 7. Note that when a congestion is detected in the channel-abstracted version, WSN-PN cannot tell exactly whether the root cause is due to collision or interference, but only able to confirm that there is possible congestion on the channels of the investigated WSN.

TABLE III: EXPERIMENTAL RESULTS

Number of Sensors	Number of Packets	Bandwidth/Buffer	Model	Property	Used memory	Total transitions	Visited states	Result
5	50	300	No Abstraction	deadlockfree	Timeout at 34837s			
				chk-channel-congestion	9185.56	125923	27940	Not valid
				chk-sensor-congestion	9582.648	158294	35320	
			Channels Abstraction	deadlockfree	12776.632	178666	37297	Valid
				chk-sensor-congestion	9740.36	3683	9008	Not
			Sensors Abstraction	deadlockfree	25829.008	531062	18045	Valid
				chk-channel-congestion	11349.368	1984	3450	Not
5	100	600	No Abstraction	deadlockfree	Timeout at 36971s			
				chk-channel-congestion	9933.424	125032	29072	Not valid
				chk-sensor-congestion	14514.352	158865	45093	
			Channels Abstraction	deadlockfree	20821.152	364759	75269	Valid
				chk-sensor-congestion	11624.52	5792	12049	Not
			Sensors Abstraction	deadlockfree	47986.464	994011	35329	Valid
				chk-channel-congestion	9741.568	3054	5299	Not
10	50	300	No Abstraction	deadlockfree	Timeout at 35558s			
				chk-channel-congestion	142192.96	5946	16230	Not valid
				chk-sensor-congestion	19821.544	5623	22256	
			Channels Abstraction	deadlockfree	167027.568	1221071	965520	Valid
				chk-sensor-congestion	11585.496	3979	1219	Not
			Sensors Abstraction	deadlockfree	117253.056	4661915	380458	Valid
				chk-channel-congestion	114344.544	1438	2209	Not
10	100	600	No Abstraction	deadlockfree	Timeout at 37628s			
				chk-channel-congestion	12940.056	59432	20093	Not valid
				chk-sensor-congestion	24823.76	1027607	35458	
			Channels Abstraction	deadlockfree	87368.256	6962674	662923	Valid
				chk-sensor-congestion	164568.32	416270	20873	Not
			Sensors Abstraction	deadlockfree	1800147.2	28519305	364272	Valid
				chk-channel-congestion	189815.616	188055	12045	Not

V. EXPERIMENTS

We conducted experiments to demonstrate the efficiency of our abstraction, which can significantly reduced the computational cost of congestion verification. The experiments were run using WSNs modelled by WSN-PN, whose numbers of sensors range from 1 to 10. The parameters of these sensors are set to enforce the congestion. Unlike other approaches using simulation, the network congestion in our experiments can be verified merely based on network topology and sensor configurations. That is, one does not need to bother the routing protocols actually used to transfer packets among the sensors. This makes our verification result still remained valid even though if the sensors are upgraded with new routing protocols in the future.

We also verify other property *deadlock-free* of the modelled WSN. For congestion checking, we separately verify the properties of *chk-sensor-congestion* or *chk-channel-congestion* (check whether the congestion occurs in Sensors or Channel or not). Table III shows our experimental results. In all cases, our abstraction leads to a decrease in the computation time and memory usage, but still guarantees the soundness of congestion verification. Some configurations could not even be analyzed with the complete model (suffering time-out status), but could using abstractions. The memory usage is also one order of magnitude smaller when using abstractions, which shows the efficiency of our approach.

At the moments, the number of sensors in simulated networks is still limited at 13, but once combined with appropriate networks as suggested in Section III.F, we can increase this number significantly. This opens an interesting direction for our future work.

The tool, the user manual, all experiments and full datasets are available on WSN-PN website[3].

VI. CONCLUSION

This paper presents WSN-PN, a tool for modelling and verifying Wireless Sensor Networks using Petri nets. WSN-PN allows for formally verifying properties such as deadlock or reachability of a WSN using model checking, and more specifically the possibility of congestion in the WSN. For a better efficiency, the tool supports abstraction of components of a WSN, focusing either on sensors or channels. It thus significantly reduces the state space generated for congestion verification, as shown in our experimental results.

Future Works: WSN-PN will be extended to verify other characteristics of WSNs, such as congestion mitigation or packet-loss recovery. We also consider using high-level

[3]http://cse.hcmut.edu.vn/~save/project/kwsn/start

Petri nets (*e.g.* coloured Petri nets [8]) to avoid network simulation by code. Time Petri nets are also a good candidate to simulate delay-sensitive events of WSN.

ACKNOWLEDGMENT

This research was partially supported by Saigon University, Ho Chi Minh City Vietnam.

REFERENCES

[1] F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] S. Moon, S. Lee, and H. Cha, "A congestion control technique for the near-sink nodes in wireless sensor networks," in *Proc. Third International Conference in Ubiquitous Intelligence and Computing*, 2006, pp. 488–497.

[3] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 215–233, 2003.

[4] C. Wan, S. B. Eisenman, and A. T. Campbell, "CODA: Congestion detection and avoidance in sensor networks," in *Proc. the 1st International Conference on Embedded Networked Sensor Systems*, 2003, pp. 266–279.

[5] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proc. the 2nd International Conference on Embedded Networked Sensor Systems*, 2004, pp. 134–147.

[6] K. Jensen and L. M. Kristensen, *Coloured Petri Nets — Modelling and Validation of Concurrent Systems*, Springer, 2009.

[7] C. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft, "Siphon: Overload traffic management using multi-radio virtual sinks in sensor networks," in *Proc. the 3rd International Conference on Embedded Networked Sensor Systems*, 2005, pp. 116–129.

[8] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proc. the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008, p. 60.

[9] Y. Si, J. Sun, Y. Liu, J. S. Dong, J. Pang, S. J. Zhang, and X. Yang, "Model checking with fairness assumptions using PAT," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 1–16, 2014.

[10] Y. Luo, L. Pu, M. Zuba, Z. Peng, and J. Cui, "Challenges and opportunities of underwater cognitive acoustic networks," *IEEE Transaction Emerging Topics Computer*, vol. 2, no. 2, pp. 198–211, 2014.

[11] M. Zheng, J. Sun, Y. Liu, J. S. Dong, and Y. Gu, "Towards a model checker for nesc and wireless sensor networks," in *Proc. 13th International Conference on Formal Engineering Methods*, 2011, pp. 372–387.

[12] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro.*, vol. 22, no. 6, pp. 12–24, 2002.

[13] Q. Shi, S. Kyperountas, F. Niu, and N. S. Correal, "Location estimation in multi-hop wireless networks," in *Proc. IEEE International Conference on Communications*, 2004, pp. 2827–2831.

Khanh Le is a lecturer in the Faculty of Information Technology, Saigon University, Vietnam. She received the bachelor degree in mathematics and computing from Natural Science HCM in 2005 and received her master degree in computer networking from Paris VI University in 2009. Now, she is a PhD student in University of Technology. Her current researches include quality of services for wireless sensor network, system modeling and system verification.

Thang Bui is currently a lecturer in the Faculty of Computer Science & Engineering, Ho Chi Minh City University of Technology, Vietnam and a member of the Laboratory for Systems Analysis and Verification (SAVE), which aims at developing automated techniques for analyzing and reasoning on computer-based systems. His research focuses on software verification, particularly model checking, which is to model the real world application and to check for any violation of desired properties. Thang holds a bachelor degree in computer engineering from Ho Chi Minh City of University of Technology in 1997, a master degree in computer science & engineering from Asian Institute of Technology, Thailand in 2001, and a PhD degree in computer science & engineering from University of New South Wales, Australia in 2010.

Tho Quan is an associate professor in the Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, Vietnam. He received his B.Eng. degree in information technology from HCMUT in 1998 and received the PhD degree in 2006 from Nanyang Technological University, Singapore. His current research interests include formal methods, program analysis/verification, the semantic web, machine learning/data mining and intelligent systems. Currently, he heads the Department of Software Engineering of the Faculty. He is also serving as the chair of Computer Science Program (undergraduate level).

Laure Petrucci has been a full professor since 2003. Her research interests concern on formal modelling and verification of complex systems, using models such as Petri nets or automata. She has a strong expertise in modular, compositional and parametric verification, in order to tackle large systems. She is currently the director of the LIPN. She received the PhD degree From University Paris 6, France in 1991. She published more than 90 publications in international conferences and journals (including CAV, Petri Nets, ATVA).