Component-Based Abstraction of Petri Net Models: An Application for Congestion Verification of Wireless Sensor Networks

Khanh Le
Ho Chi Minh City University of Technology
Vietnam
lnkkhanh@cse.hcmut.edu.vn

Thang Bui
Ho Chi Minh City University of Technology
Vietnam
thang@cse.hcmut.edu.vn

Tho Quan
Ho Chi Minh City University of Technology
Vietnam
qttho@cse.hcmut.edu.vn

Laure Petrucci
LIPN, CNRS UMR 7030
Université Paris 13
Sorbonne Paris Cité
Villetaneuse, France
Laure.Petrucci@lipn.fr

Étienne André
LIPN, CNRS UMR 7030
IRCCyN, CNRS UMR 6597
Villetaneuse / Nantes
France
Etienne.André@lipn.fr

ABSTRACT

This paper proposes a new approach for modelling a Congestion Detection mechanism on Wireless Sensor Networks (WSN) using the Petri Net (PN) language. Even though PNs are powerful for modelling concurrent systems, they suffer from a high computational cost when verifying properties on the modelled system. We suggest a component-based abstraction to tackle this problem. First, a PN system can be considered as a set of connected *components*. For instance, a WSN includes a *Sensor component* and a *Channel component*. When performing verification on a model, we can abstract the insignificant components and only focus on the remaining ones. Experiments demonstrate that our approach is practically feasible when tested with various kinds of congestion occurring on different PN modelling WSNs.

Keywords

Wireless Sensor Networks, Petri Nets, Compositional Verification, Abstraction

CCS Concepts

•**General and reference → Verification;**

1. INTRODUCTION

The Petri Net (PN) formalism (*e.g.* [1]) is a graphical mathematical language which efficiently supports the modelling and verification of distributed systems. Basically, a Petri net is a directed bipartite graph, featuring transitions and places. Each *place* contains a number of *tokens*. When all source *places* of a *transition* have enough requested *tokens*, this *transition* is *firable*. Whenever the *transition* is *fired*, the requested *tokens* are removed from the source *places* and new *tokens* are created in destination *places*.

As PNs are widely used in research and industry communities, there are several tools developed to help users specify and verify Petri nets, in particular Snoopy [2], TAPAAL [3], *CosyVerif* [4] or CPN Tools [5]. Although most tools work with basic place/transitions PNs, some of them handle high-level PNs such as timed, coloured, or stochastic PNs.

When a system is modelled by a Petri net, its properties can be verified using model checking techniques. However, the biggest disadvantage of this approach is the famous state space explosion problem. There are two main approaches to cope with it: reducing the state space or abstracting the PN model. In this paper, we focus on the latter.

Several approaches to Petri net abstraction have been proposed based on *compositional or modular Petri nets* [6, 7, 8]. In such a model, a PN includes several modules which have some transitions that must be fired synchronously, known as *synchronised transitions*. Based on the analysis of synchronised transitions, one can abstract and verify each module in an incremental manner [9].

However, this approach has some drawbacks. When modelling a real system as a PN, one needs to capture all semantic actions of the system, which are usually described by a domain expert, as semantic rules on the corresponding PN model. Furthermore, when abstracting such a PN model, it is also necessary to ensure that these semantic rules still hold. This is not trivial, since the abstraction is usually performed based on the internal relationship and structure of the model, not taking into account the semantics of the real system.

In addition, in some applications, the system to be modelled by a PN is made from several *basic components*, each of which can be considered as a module [10]. Hence, the number of modules in a system is sometimes very large, corresponding to the real components integrated in the system. However, these components can be classified into a finite

SoICT 2015, December 03-04, 2015, Hue City, Viet Nam
© 2015 ACM. ISBN 978-1-4503-3843-1/15/12…$15.00
DOI: http://dx.doi.org/10.1145/2833258.2833298

number of *types*. For instance, a Wireless Sensor Network (WSN) is a collection of sensors and their connections, *i.e.* channels. On the other hand, when the system is verified, we only need to focus on some components and discard the others, according to the properties to be verified.

This observation prompts us to propose a *component-based Petri net* approach. Thus, a PN model will be constituted by some *component PNs*. Each component PN can be considered as sub-PN model that represents a physical component in the real system. Once abstracted, each component PN is reduced into an abstracted one. Therefore, the original PN model and its abstracted model are always equivalent to the real system w.r.t. the communicating components. Hence, semantic rules specified by domain experts for the operational mechanism of the components in the real system can still be applicable for both original and abstracted PN models.

In this paper, we also illustrate the advantage of this approach by an application to congestion detection in Wireless Sensor Networks. The two main components of WSNs to consider are sensors and channels.

Once these components are abstracted properly, the state space is significantly reduced, while maintaining the correctness of congestion verification by using the same semantic rules as for the original WSN.

Outline. The rest of the paper is organised as follows. Section 2 discusses related works. Section 3 introduces the WSNs and the congestion detection problem. Section 4 proposes our novel model for abstraction and Section 5 applies it to our case study to detect congestion in a WSN. More extensive experiments are reported in Section 6. Finally, Section 7 draws conclusions and outlines future work.

2. RELATED WORK

2.1 Network modelling using Petri Nets

Petri nets are particularly well-suited for modelling and analysing network systems and protocols [11]. In [12] a model is proposed to implement ATM (Asynchronous Transfer Mode) networks and the applications running over it. Transferring all multimedia data over the switching network of an ATM network constitutes a big challenge. Since ATM is a connection oriented protocol, the switching network must establish a virtual connection from one of its input ports to an output port before forwarding incoming ATM cells (packets) along that virtual connection. Because of restricted bandwidth, multimedia data must be split to fix-length units before sending without losing behaviour. Most approaches use queue-based or stochastic-based models, but however ignore synchronisation. Dividing the ATM network into some synchronous models and forcing them to cooperate increases the transmission rate. Using Petri nets for modelling has proved promising.

[13] uses Petri nets to model a LAN switched network architecture. Components of this model include switches, servers, clients and interaction between them. The purpose of this model is to verify the influence of the switch buffer size and the rate of packets loss on the quality of the transmission.

Stochastic Petri nets are also used to model and analyse ad-hoc wireless networks in [14]. Ad-hoc wireless networks are dynamic networks with mobile nodes whose bandwidth and battery are limited.

2.2 Model Checking and Compositional Verification on Abstracted Models

Model checking (*e.g.* [15]) allows for analysing complex concurrent systems in order to investigate their behaviour and verify whether a specified property holds during the system execution. This task is achieved by exploring the state space, *i.e.* a directed graph whose nodes represent reachable states of the system and the arcs the transitions between states.

Nowadays, model verification has attracted much attention of many researchers and communities. For instance, the SLAM project [16] applies the model verification idea to verify microprocessor *Intel Core i7*. However, the disadvantage of state space explosion became a serious problem when applying model verification to industrial systems. Therefore, much research has been proposed in order to solve this problem. Most of them focus on the techniques of abstraction and decomposition, which is considered as *compositional verification.*

In this approach, the architecture of the original system can be divided into sub-systems. The strategy of verification for this system is thus based on sub-system verification [17]. The general properties of the system can be proved based on the properties of its sub-systems. The verification process includes the following steps: verification of sub-systems, analysis on the relationship of sub-systems, combining the local properties of all sub-systems to global properties for the whole system.

Modular verification (*e.g.* [7]) is an efficient method based on the design of the system. Firstly, the state space is decomposed into two parts: sub-systems and a graph to synchronise these sub-systems. A decomposition approach [18] divides the system into concurrent sub-systems. Each sub-system has its own state space. The full state space of the system is then established by the state spaces of the sub-systems or the sub-systems' actions only. This proposal is suitable for synchronous systems.

Furthermore, abstraction techniques have recently been proposed. Basically, abstraction reduces a PN model into a new one which has a smaller size in terms of number of places and transitions. As a trade-off, the abstracted model cannot preserve all properties from the original one, but only some needed properties [19, 20, 21]. Thus, we cannot ensure the soundness nor completeness when verifying the abstracted model against certain properties. For instance, in the work of [22], if the abstracted model satisfies a property *e.g. deadlock*, the original model satisfies it too, but the converse is not true. Other proposals [23, 8] use counter-examples as a key for compositional verification of a system combined from multiple modules. If a counter-example is found in an abstracted module, this counter-example is verified again on the corresponding concrete module. Otherwise, *i.e.* no counter-example found on any abstracted module for a given property, the real system does indeed satisfy the property.

3. CONGESTION DETECTION IN WIRELESS SENSOR NETWORKS

3.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are networks that consist of hundreds or thousands of *sensors*, which are small, cheap, lower-power and lower-energy equipments. Each sensor includes sensing, data processing and communicating components (see *e.g.* [24]). Sensors communicate with one another via wireless signals when deployed close enough, and thus transmit data. Hence, generally, WSNs include a set of sensors and their communication *channels*. Sensors can be divided into three kinds: sources, sinks and intermediate sensors. Sources are initial sensors, *i.e.* sensors which generate packets and send them to their neighbours, while sinks can be considered as processing sensors, *i.e.* they receive packets from the other sensors and process them. Intermediate sensors are special sensors: they receive packets from neighbours and forward them to target nodes without interacting or processing packets. When two sensors are able to exchange information with each other, we say that there is a *channel* established between them.

An example of a network topology is given in Fig. 1, where the circles depict the wireless range of the sensors, and the lines between them make the communication channels explicit.

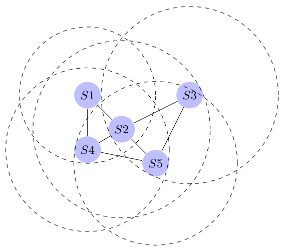

Figure 1: Wireless Sensor Network Example.

WSNs are deployed according to a dense or a sparse mode to cover a lot of applications. Environment systems which are 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 [25]. 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 [26].

3.2 Congestion in WSNs

Although convenient, using WSNs raises several challenges to handle their constraints. These are mainly of two types: *sensor constraints* and *network constraints* [27]. Checking whether a WSN satisfies certain constraints is a challenging issue. A difficult task relating to constraints verification, which has attracted much attention, is *congestion handling*, which generally comprises two main phases: *congestion detection* and *congestion mitigation.* In this paper, we focus on congestion detection.

The transportation of data packets over the network leads to varying degrees of congestion in WSNs. Congestion can be detected in both dense and sparse deployments. 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 channels in sparse mode due to interference [28].

In a popular research of WSN congestion reported in [28], there are three typical scenarios of congestion, which are recalled as follows.

- **Scenario 1 (dense deployment and high transmission rate)** The network is simulated based on disaster-related-event (fire or earthquake) occurring during 10 seconds. The model has 30 nodes and each node generates at least 100 packets to the sinks. At the sinks, within one interval (500 ms), the number of packets dropped is counted to detect the appearance of congestion in the network. Since the network deployment is dense, *i.e.* sensors are very close, some packets may collide as they are transmitted over the same paths. The congestion in this case occurs in channels. Moreover, the speed for sending packets in the network may be too high, *i.e.* the processing rate of sensors may be smaller than their receiving rate. So, the network can be subject to congestion due to sensors' buffer overload.

- **Scenario 2 (sparse deployment and low transmission rate)** The network is deployed with 15 nodes and the rate for sending packets is at most 20 packets per second. Open-loop control turns out to be limited in large networks and is thus replaced with a close-loop control mechanism. Thanks to both the long range of sensors and the low speed, the channels have enough time to transmit packets, so this control is just focusing on congestion detection in some sensors.

- **Scenario 3 (sparse deployment and high transmission rate)** The rate for transmitting is 50 packets per second over a 30 seconds simulation time. High transmission rate and long distance in this case can make the channels a bottleneck. So congestion can occur in the channels. To reduce congestion, both open-loop and close-loop control mechanisms are applied.

3.3 Congestion Detection Algorithms and Tools

Most congestion detection algorithms use a buffer/queue as main key for computation. Siphon [29] 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) [28], uses both buffer threshold, buffer weight for detection and combines with bottleneck-node-based method to control the sending and reception of packets. In Fusion [30], 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 take it into account and stop forwarding packets to the congested node so that it can drain the backlogged packets.

To understand congestion detection activity, most algorithms were simulated in 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 [31] or Omnet++ [32]. In these simulators, a WSN is modelled not only by its sensors and channels, but also by the frameworks used to support the *routing protocol.* For example, in Omnet++, the `mf` framework was used first, and then changed to the `inet` framework. It means that the users must completely change all their predefined models to reflect the change of routing protocol.

Discussion. Even though simulators are practically useful and currently widely used, the simulation analysis may miss some scenarios. The reason is that in order to capture all of possible scenarios, a user must accordingly set all suitable parameters for the tool and observe the resulting simulation. Thus, certain practical situations may be missed, especially those with small probability.

In this paper, we use a different approach to detect congestion. First, a WSN is modelled by a PN, based on its characteristics and rules for congestion detection are expressed in LTL (Linear Temporal Logic, see *e.g.* [15]). 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.

However, the drawback of this approach is the famous state space explosion problem. Indeed, the state space, describing all possible behaviours of the system, can be very large, or even infinite. Without further techniques, it may be impossible to explore the entire state space with limited resources in time and memory. Note that, this is a trade off between a completeness of verification and incompleteness of simulation.

Therefore, in order to reduce the state space, without loss of behaviour w.r.t. congestion verification, we apply the component-based abstraction approach. Based on the three congestion scenarios presented above, there are various approaches to detect congestion in a WSN, depending on its characteristics. In order to verify if the WSN is congested due to buffer overload, we only need to check the status of sensors. Conversely, if we want to focus on packets collision or interference, only channels are to be inspected. In our approach of Component-based PN model of WSN, every element in a WSN (*i.e.* sensor and channel) can be modelled as a *component*, and can also be abstracted if it is not a cause of congestion. Since the abstracted components are virtually not needed to be explored in the verification process, the state space exploration thus can be reduced. As a result, our approach can verify some real WSN settings, which the conventional formal approach fails to handle due to the state space explosion.

4. COMPONENT-BASED ABSTRACTION

We first recall the definition of Petri nets and introduce components.

Definition 1 (Petri net).
A Petri net *is a tuple* $\mathcal{N} = \langle P, T, F, W, M_0 \rangle$ *where:*

- P *is a finite set of* places.
- T *is a finite set of* transitions.
- P *and* T *are disjoint, i.e.* $P \cap T = \emptyset$.
- $F \subseteq (P \times T) \cup (T \times P)$ *is a* flow *relation.*
- $W : F \to (\mathbb{N} \setminus 0)$ *is an* arc weight *mapping.*
- $M_0 : P \to \mathbb{N}$ *is the* initial marking.

Definition 2 (Component).
A component *Com is a Petri net where the set of places is the union of three disjoint sets:* $P_{Com} = P_{in} \uplus P_{out} \uplus P_{inter}$ *defined as follows:*

- P_{in} *is a non-empty set of* input places*, i.e.* $(T \times P_{in}) \cap F = \emptyset$.
- P_{out} *is a non-empty set of* output places*, i.e.* $(P_{out} \times T) \cap F = \emptyset$.
- P_{inter} *is the set of all other (intermediate) places in Com.*

For instance, a sensor in a WSN can be modelled as a component as shown in Fig. 2 where *P_in* is the input place, *P_out* the output place and *P_inter* an intermediate place. Transition *Generate Packets* allows for the *Sensor* to generate new packets and then send them with transition *Send Packets.* Similarly, a *Channel* is modelled as shown in Fig. 3, which has two main functions: receiving packets (transition *Receive Packets*) and sending packets (transition *Send Packets*).

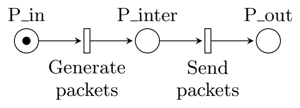

Figure 2: Source Sensor Component.

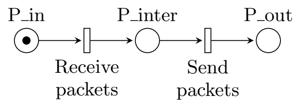

Figure 3: Channel Component.

Definition 3 (Component-based Petri net).
A component-based Petri net *is a Petri net made from a disjoint set of components* $S_{Com} = \{Com_1, \ldots, Com_n\}$ *and a set of* connectors $S_C = \{C_1, .., C_k\}$ *where a connector* C_i *is a transition from the output places of a component to the input places of another component, as depicted in Fig. 4.*

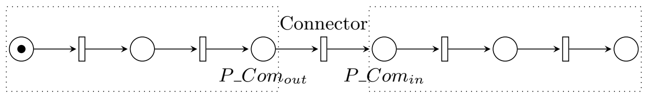

Figure 4: Connector.

Fig. 5 is an example of a Component-based Petri net which includes three components *Source* (S_1), *Channel* (T_1) and *Sink* (S_2) and two connectors T1_con and T2_con. We hereafter refer to this Petri net as *WSNCom.*

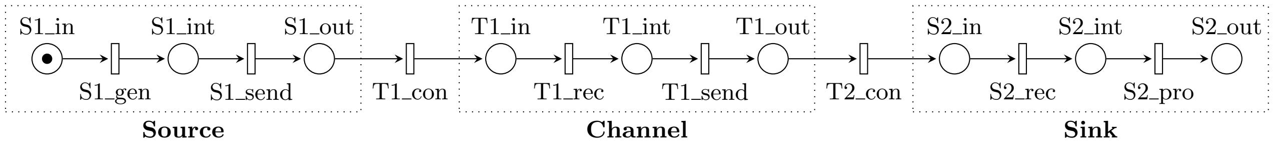

Figure 5: Component-Based PN of Wireless Sensor Network.

Definition 4 (Typed Component-based Petri net). *A component-based Petri net with component set S_{Com} is said to be* typed *if $\exists f_{type} : S_{Com} \rightarrow \mathfrak{D}$ where $\mathfrak{D}$ is a finite concrete set called* Component Domain.

For example, the component-based PN *WSNCom* is typed, with function f_{type} such that:
$\mathfrak{D} = \{Sensor, Channel\}$
$f_{type}(S_1) = Sensor$
$f_{type}(S_2) = Sensor$
$f_{type}(T_1) = Channel$
Intuitively, one can consider that there are two types of components in a WSN: *sensors* and *channels*.

Definition 5 (Component Abstraction). *A component Com in a Component-based PN can be reduced to an* abstracted place *(*place-based abstraction*). Alternatively, Com and all of its attached connectors can be reduced to an* abstracted transition *(*transition-based abstraction*).*

Definition 6 (Type-Abstraction Function). *Let PN-Com be a component-based Petri net and $\mathcal{J}$ a value of Component Domain $\mathfrak{D}$. The Type-Abstraction function creates an abstracted Petri net $Com_{\mathcal{A}}$ from PN-Com by replacing all components typed $\mathcal{J}$ in PN-Com by its corresponding abstracted component, denoted as $Com_{\mathcal{A}} = \mathcal{J}^{\langle p \rangle}$PN-Com if the abstraction is place-based, or $Com_{\mathcal{A}} = \mathcal{J}^{\langle t \rangle}$PN-Com if the abstraction is transition-based.*

Fig. 6 is an example of $Sensor^{\langle p \rangle}(WSNCom)$. All components typed *Sensor*, *i.e.* S_1 and S_2, have been abstracted as abstracted places S_1 and S_2, depicted larger and dashed.

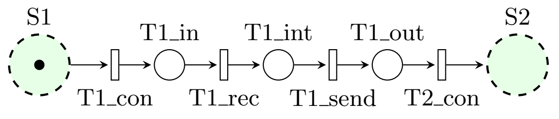

Figure 6: Sensor Abstraction.

Moreover, Fig. 7 shows $Channel^{\langle t \rangle}(WSNCom)$, where component typed *Channel* T_1 and its two connectors ($T1_{con}$ and $T2_{con}$) have been abstracted into a new abstracted transition *T_a*, depicted larger and dashed.

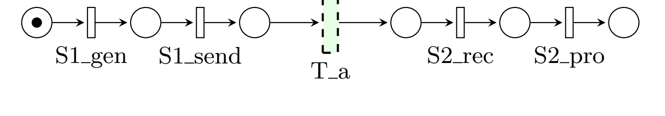

Figure 7: Channel Abstraction.

Finally, Fig. 8 represents the Petri net resulting from $Sensor^{\langle p \rangle}(Channel^{\langle t \rangle}(WSNCom))$. In fact, it is the network topology of the original WSN represented by *WSNCom*.

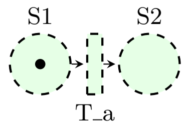

Figure 8: Result of $Sensor^{\langle p \rangle}(Channel^{\langle t \rangle}(WSNCom))$.

Remark. The congestion property is preserved under **Type-Abstraction Function**. For instance, in reality, the congestion occurs on Channel. So, the abstraction model is established based on Sensor Abstraction, *i.e.* congestion property still appears on Channel. Therefore, the result of congestion detection is the same in both real and abstracted models.

5. CASE STUDY: PROPOSED MODELS FOR WIRELESS SENSOR NETWORKS

In this section, we model a WSN using a component-based Petri net as discussed in Section 4 and use a component-abstraction approach to detect congestion in an efficient way. For this purpose, we developed a tool WSN-PN which helps users to specify the topology of a WSN from an interface layer. Then, the tool automatically generates a corresponding component-based PN and verify the congestion property using the PAT model checking library [33]. The tool, the user manual, all experiments and full datasets are available at http://cse.hcmut.edu.vn/˜save/project/kwsn/start.

5.1 WSN models

An example of network topology for a WSN is presented in Fig. 9. We assume all packets are sent via broadcast. Fig. 10 shows the Petri net model of our WSN example for Congestion Detection in Sensor Nodes and Channels. (All PN models in Figs. 10 to 12 are automatically generated by WSN-PN, which explains the layout.)

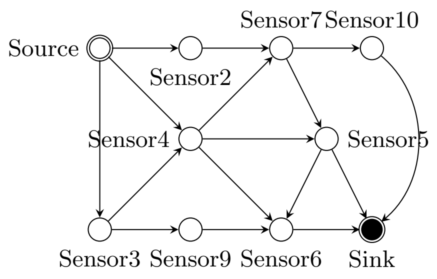

Figure 9: Wireless Sensor Network Topology.

This WSN can be abstracted based on its channels or

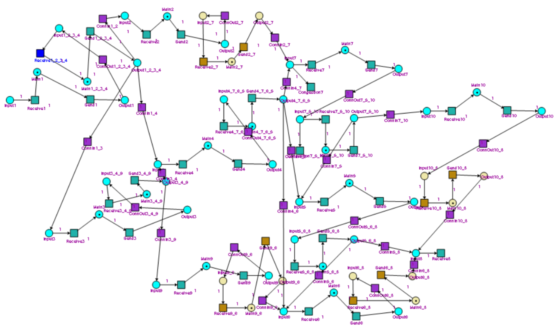
Figure 10: Model for Congestion Detection in Sensor Nodes and Channels.

sensors, resulting in the abstracted Petri nets presented in Figs. 11 and 12 respectively.

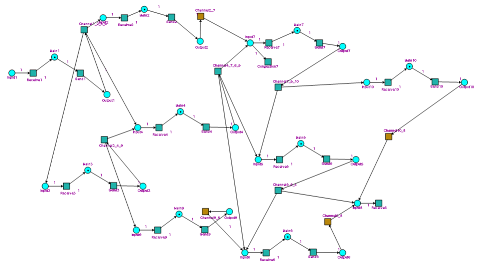
Figure 11: WSN Model in Fig. 10 channel-abstracted.

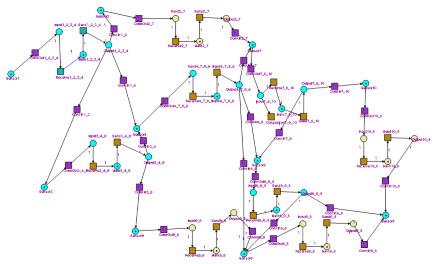
Figure 12: WSN Model in Fig. 10 sensor-abstracted.

5.2 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. Each channel is also encoded as a pair $\{B_c, t\}$, where B_c is 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.* send out to connected sensors). The operational semantic rules of the encoded sensors and networks are presented in Table 1.

As discussed, each sensor and channel is modelled as 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.

6. EXPERIMENTATIONS

We use a server with 2.5 Ghz CPU and 24 GiB memory for our verification. Table 2 shows the initial parameters for experimenting. (Most of these initial parameters are those from CODA's experimentations.)

Table 2: Initial Parameters

Parameter	Range
Number of Sensor Nodes	5-10
Number of packets	30-100
Sensor buffer size	200-600
Channel bandwidth (buffer)	200-600
Sending rate	2-3 packets/ms
Processing rate	1-2 packets/ms

We verified the following properties using WSN-PN:

- *deadlock-free*: whether a deadlock occurs.
- *chk-sensor-congestion*: whether congestion occurs in sensors.
- *chk-channel-congestion*: whether congestion occurs in channels.

Table 3 shows our experimental results. In all cases, our abstraction leads to a decrease in the computation time and memory usage. Some configurations could not even be analysed with the complete model, but could using abstractions. The memory usage is also one order of magnitude smaller when using abstractions, which shows the efficiency of our approach.

7. CONCLUSION

In this paper, we proposed an approach for Congestion Detection for WSN models for both dense and sparse deployments.

Using the Petri net model, it is possible to check whether a congestion occurs in sensor nodes or in channels or both of them. The use of appropriate abstracted models reduces the state space explosion and thus allows for pushing further the limits of verification. Our experiments using our tool WSN-PN show that the state space of the sensor-abstracted or

Table 1: 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
$\frac{\{B,Q,p,s\},B>0}{B=max(0,B-\frac{1}{p}),Q=Q+\frac{1}{p}}$ *[sensor-processing]*	This rule is applied when a packet is internally processed within a sensor

Table 3: 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	Not valid
			Channels Abstraction	*deadlockfree*	12776.632	178666	37297	Valid
				chk-sensor-congestion	9740.36	3683	9008	Not valid
			Sensors Abstraction	*deadlockfree*	25829.008	531062	18045	Valid
				chk-channel-congestion	11349.368	1984	3450	Not valid
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	Not valid
			Channels Abstraction	*deadlockfree*	20821.152	364759	75269	Valid
				chk-sensor-congestion	11624.52	5792	12049	Not valid
			Sensors Abstraction	*deadlockfree*	47986.464	994011	35329	Valid
				chk-channel-congestion	9741.568	3054	5299	Not valid
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	Not valid
			Channels Abstraction	*deadlockfree*	167027.568	1221071	965520	Valid
				chk-sensor-congestion	11585.496	3979	1219	Not valid
			Sensors Abstraction	*deadlockfree*	117253.056	4661915	380458	Valid
				chk-channel-congestion	114344.544	1438	2209	Not valid
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	Not valid
			Channels Abstraction	*deadlockfree*	87368.256	6962674	662923	Valid
				chk-sensor-congestion	164568.32	416270	20873	Not valid
			Sensors Abstraction	*deadlockfree*	1800147.2	28519305	364272	Valid
				chk-channel-congestion	189815.616	188055	12045	Not valid

channel-abstracted models is significantly reduced compared to the one of the original model, and the computation time can be decreased by one order of magnitude.

In the future, we will extend this approach to *time* Petri net models, so as to model the different timings in WSNs such as a randomised time for sending packets or a processing time by the sensor.

Acknowledgements

This research was partially supported by Saigon University, Ho Chi Minh City Vietnam, and by STIC-Asie "CATS" (Compositional Analyses of Timed Systems) project.

8. REFERENCES

[1] V. E. Kozura, V. A. Nepomniaschy, and R. M. Novikov, "Verification of distributed systems modelled by high-level Petri nets," in *2002 International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, 2002, pp. 61–66.

[2] M. Heiner, R. Richter, and M. Schwarick, "Snoopy: a tool to design and animate/simulate graph-based formalisms," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SimuTools 2008)*, 2008, p. 15.

[3] J. Byg, K. Y. Jørgensen, and J. Srba, "An efficient translation of timed-arc Petri nets to networks of timed automata," in *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods (ICFEM 2009)*, 2009, pp. 698–716.

[4] É. André, Y. Lembachar, L. Petrucci, F. Hulin-Hubard, A. Linard, L. Hillah, and F. Kordon, "*CosyVerif*: An open source extensible verification environment," in *18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*, 2013, pp. 33–36.

[5] M. Westergaard and T. Slaats, "CPN tools 4: A process modeling tool combining declarative and imperative paradigms," in *Business Process Management: 11th International Conference (BPM 2013)*, 2013.

[6] D. Chiarugi, P. Degano, and R. Marangoni, "A computational approach to the functional screening of genomes," *PLoS Computational Biology*, vol. 3, no. 9, 2007.

[7] C. Lakos and L. Petrucci, "Modular analysis of systems composed of semiautonomous subsystems," in *Proc. 4th Int. Conf. on Application of Concurrency to System Design (ACSD'04), Hamilton, Canada, June 2004*, Jun. 2004, pp. 185–194.

[8] É. André, H. Ochi, K. Klai, and L. Petrucci, "A counterexample-based incremental and modular verification approach," in *17th Monterey Workshop on Development, Operation and Management of Large-Scale Complex IT Systems*, ser. Lecture Notes in Computer Science, R. Calinescu and D. Garlan, Eds., vol. 7539. Oxford, England: Springer, Aug. 2012, pp. 283–302.

[9] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre, "Incremental verification by abstraction," in *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, (TACAS*

2001), 2001, pp. 98–112.

[10] P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin, “Petri nets compositional modeling and verification of flexible manufacturing systems,” in *IEEE Conference on Automation Science and Engineering (CASE 2011)*, 2011, pp. 588–593.

[11] J. Billington, G. R. Wheeler, and M. C. Wilbur-Ham, “PROTEAN: A high-level Petri net tool for the specification and verification of communication protocols,” *IEEE Transactions on Software Engineering*, vol. 14, no. 3, pp. 301–316, 1988.

[12] M. Reid and W. M. Zuberek, “Timed Petri net models of ATM LANs,” in *Application of Petri Nets to Communication Networks, Advances in Petri Nets*, ser. Lecture Notes in Computer Science, vol. 1605. Springer, 1999, pp. 150–175.

[13] D. A. Zaitsev, “Switched LAN simulation by colored Petri nets,” *Mathematics and Computers in Simulation*, vol. 65, no. 3, pp. 245–249, 2004.

[14] C. Zhang and M. Zhou, “A stochastic Petri net-approach to modeling and analysis of *ad hoc* network,” in *Information Technology: Research and Education (ITRE 2003)*. IEEE, 2003, pp. 152–156.

[15] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[16] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodová, C. Taylor, V. Frolov, E. Reeber, and A. Naik, “Replacing testing with formal verification in Intel CoreTM i7 processor execution engine validation,” in *Computer Aided Verification, 21st International Conference, (CAV 2009)*, 2009, pp. 414–429.

[17] E. M. Clarke, D. E. Long, and K. L. McMillan, “Compositional model checking,” in *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS ’89)*, 1989, pp. 353–362.

[18] A. Valmari, “Compositionality in state space verification methods,” in *Application and Theory of Petri Nets (APN 1996)*, 1996, pp. 29–56.

[19] S. Haddad, J. Ilié, and K. Klai, “Design and evaluation of a symbolic and abstraction-based model checker,” in *Automated Technology for Verification and Analysis: Second International Conference (ATVA 2004)*, 2004, pp. 196–210.

[20] K. Klai and D. Poitrenaud, “MC-SOG: an LTL model checker based on symbolic observation graphs,” in *Applications and Theory of Petri Nets, 29th International Conference (PETRI NETS 2008)*, 2008, pp. 288–306.

[21] A. Duret-Lutz, K. Klai, D. Poitrenaud, and Y. Thierry-Mieg, “Self-loop aggregation product - A new hybrid approach to on-the-fly LTL model checking,” in *Automated Technology for Verification and Analysis. 9th International Symposium (ATVA 2011)*, 2011, pp. 336–350.

[22] K. Klai, S. Haddad, and J. Ilié, “Modular verification of Petri nets properties: A structure-based approach,” in *Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, 2005, pp. 189–203.

[23] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement for symbolic model checking,” *ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[24] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[25] S. Moon, S. Lee, and H. Cha, “A congestion control technique for the near-sink nodes in wireless sensor networks,” in *Third International Conference on Ubiquitous Intelligence and Computing (UIC 2006)*, ser. Lecture Notes in Computer Science, vol. 4159. Springer, 2006, pp. 488–497.

[26] 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.

[27] D. W. Carman, P. S. Kruus, and B. J. Matt, “Constraints and approaches for distributed sensor network security (final),” DARPA Project report (Cryptographic Technologies Group, Trusted Information System, NAI Labs), Tech. Rep., 2000. [Online]. Available: https://www.csee.umbc.edu/courses/graduate/CMSC691A/Spring04/papers/nailabs_report_00-010_final.pdf

[28] C. Wan, S. B. Eisenman, and A. T. Campbell, “CODA: congestion detection and avoidance in sensor networks,” in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM, 2003, pp. 266–279.

[29] C. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft, “Siphon: overload traffic management using multi-radio virtual sinks in sensor networks,” in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys 2005)*. ACM, 2005, pp. 116–129.

[30] B. Hull, K. Jamieson, and H. Balakrishnan, “Mitigating congestion in wireless sensor networks,” in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*. ACM, 2004, pp. 134–147.

[31] “The Network Simulator NS-2,” http://www.isi.edu/nsnam/ns/.

[32] A. Varga and R. Hornig, “An overview of the OMNeT++ simulation environment,” in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SimuTools 2008)*, 2008, p. 60.

[33] 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.