*International Journal of Applied Information Systems (IJAIS) ï£¡ ISSN : 2249-0868*
*Foundation of Computer Science FCS, New York, USA*
*Volume 11 - No. 7, December 2016 - www.ijais.org*

# Modelling and Congestion Detection of Wireless Sensor Networks: A Concurrent-based Approach using Coloured Petri Nets

Giang Trinh Khanh Le Tam Bang Quan Tram Thang Bui Tho Quan
Hochiminh City University of Technology
Hochiminh City, Vietnam

## ABSTRACT

Congestion in Wireless Sensor Networks (WSNs for short) causes not only packet loss and but also leads to excessive energy consumption. Therefore, congestion needs to be detected as well as controlled in order to prolong system lifetime. There are two streams to concern congestion detection including simulation-based and model-based. Following the second stream, formal modelling techniques are used for analysis of WSNs. Coloured Petri nets (CPNs for short) that combines Petri nets with programming languages is a powerful modelling technique. This paper presents a CPN-based approach for formal modelling and congestion detection of WSNs. The proposed model describes parameters and behaviours of a WSN. Then the congestion detection problem is reduced to a reachability problem on the state space of the CPN-based model. Moreover, the CPN-based model uses the hierarchical modelling capability of CPNs, including different levels of abstraction (sub-modules). This helps easily handling and extending the model.
In reality, WSN components (sensors and channels) can execute a number of concurrent operations. This is called concurrency of WSNs. The CPN-based model is extended to express the concurrency, thus improving the congestion detection results.

### General Terms:

System Modelling, System Verification

### Keywords

Wireless Sensor networks, Congestion detection, Concurrency architecture, Petri nets, Coloured Petri nets

## 1. INTRODUCTION

A Wireless Sensor Network (WSNs for short) is a collection of hundreds or thousands of *sensor nodes* (*sensors* for short). Sensors are characterised as cheap, low energy consuming, with limited memory and processing capabilities [1]. Basically, sensors in a WSN can communicate with one another using WiFi signals. There are three types of sensors: *source*, *sink* and *intermediate*. 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, a *channel* is established between them. Information on sensors and channels form the topology of a WSN.

WSNs have distributed on different applications such as military target tracking and surveillance, biomedical health monitoring [2]. Although convenient, using WSNs raises several challenges since their constraints must be handled (e.g. energy consumption, transmission rate, memory, location sensing, unreliable communications, population density). Checking whether a WSN satisfies certain constraints is a challenging issue. A difficult task called *congestion handling* has attracted much attention [3, 4, 5, 6]. The task relates to constraints verification and includes congestion detection and congestion mitigation. [3] shows that congestion occurs if *buffer overload*, i.e. the rate of incoming packets is large than its processing. This paper focuses on congestion detection based on buffer overload.

Nowadays, there are two main approaches to address congestion detection including simulation-based [7, 8] and model-based [4].

In the first case, a simulator is used to mimic the operations of the WSN, measure the performance, and check congestion occurs or not. Widely used simulators include ns-2 [7] and Omnet++ [8]. In these, a WSN is considered as a network with sensors, channels and their activity (protocols). The activity of protocols in network is programmed based on the help of simulator frameworks. For example, in Omnet++, the `mf` framework was used first, and then changed to the `inet` framework.

In the second case, the model-based approach shows two immediate advantages against simulator approaches:

(1) 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, whatever the framework being used.
(2) verifies the properties by a logic formula via model checking techniques while simulators must be done by programming, thus this approach may define all scenarios.

Formal modelling techniques have proved the advantages for analysis of WSNs. To the best of our knowledge, [4] is the sole framework so far supports both modelling and congestion detection of WSNs by Petri nets. However, [4] used Place/Transition nets which is classical Petri nets with limited modelling powerful, thus existing following drawbacks. First, parameters of a sensor is expressed by variables embedded in the analysis tool. Second, the semantics of WSN operations are expressed by using code program associated to transitions. These code are in form of C# language, thus forcing the human modeller must get meanings of both Petri nets and C# language. Moreover, the human modeller also difficultly maps components of the modelled PN to corresponding parts of these code. These drawbacks force us remodel WSN by using other advanced kinds of Petri nets, Coloured Petri nets (CPNs for short) [9]. CPNs that combines Petri nets with programming languages is a powerful modelling technique. CPNs also has been applied widely for modelling and analysis of complex systems, especially networks [9, 10, 11, 12, 13]. It provides parameterised representations which allow easily keeping and handling all parameters of WSN components without interaction to code on every transition. Therefore, this paper focuses on using CPNs for modelling of Wireless Sensor networks and detection of congestion on the CPN modelled WSN. Such model has the following objectives:

—representing all WSN components and their operations;

—defining a clear operation semantics of the modelled CPN;

—supporting an easy method to detect congestion.

In reality, the sensor node executes a number of concurrent operations: generating packets (capturing information), processing, and sending packets; while the channel concurrently receives and transmits packets [14]. Moreover, a component (sensor or channel) can operate independently with each others. So operations of sensors and channels in a WSN should be performed concurrently. This is called concurrency of WSNs. Since all tools supporting Petri nets implemented only sequential semantic of firing rule, it is too difficult to express the concurrency of WSNs on the CPN modelled WSN. Therefore, the model is extended by changing semantic of firing rule. This makes behaviour of the model becomes more realistic, thus improving the congestion detection results.

CPN Tools[1] is the most popular and powerful tool for constructing and analysing CPNs. Especially, it supports timed colour set, hierarchical net. It can investigate the behaviour of the modelled system using simulation. It can also verify properties by means of state space methods and model checking [15]. Therefore, CPN Tools is used to implement the proposed approach.

### 1.1 Related Work

First, we consider generally some network modelling approaches based on Petri nets.

Petri nets is used to model a LAN switched network architecture [16]. This model includes three components (switches, servers, and clients) and interaction between these components. The influence of the switch buffer size is verified. The rate of packets loss on the quality of the transmission is also considered. [17] uses Stochastic Petri nets to model and analyse ad-hoc wireless networks.

Then, we consider particularly modelling of WSNs based on Petri nets.

There are many approaches which have used PNs and its extensions to model and analyse protocols in WSNs. Coloured Petri nets (CPNs) that is an extension of PNs is a powerful modelling technique. In [18], a CPN model is presented for modelling and performance evaluation of a medium access control protocol in WSNs named sensor-medium access control protocol (S-MAC). The proposed model for this protocol uses hierarchical CPNs (HCPNs), which extends CPNs with modules. In [13], a HCPN model is exploited to analyse the behaviour for one of the routing protocols (Vector Based Forwarding) in WSNs. Verification of this model is performed by analysing the state space statistics. These statistics include liveness, responsiveness, and free from deadlocks.

The authors of [11] use generalised stochastic Petri nets (GSPNs) to describe the data flow of WSN. The obtained model is used to analyse the impact of data aggregation on network latency and consequently in sensor's battery life.

In [19] authors propose an approach for evaluating the WSN lifetime by simulating its power consumption using CPNs. This approach includes three main parts such as a fully automated process for evaluating the WSN lifetime, a set of reusable CPN models expressing the power consumption of WSN communication protocols, and a strategy for composing CPN models of WSN applications and protocols.

[1] `http://cpntools.org/`

[12] presents a CPN-based model using hierarchical CPNs. The model can present a large class of WSN behaviour by its generality. The modelled behaviour includes monitoring and collecting data, accessing and evaluating the information, formulating meaningful user displays, and performing decision making and alarm functions. The CPN-based model is verified both quantitative and qualitative properties.

Finally, we consider both modelling and congestion detection of WSNs based on Petri nets.

To the best of our knowledge, [4] is a unique approach for both formal modelling and congestion detection of WSNs. Dealing with the problem of congestion detection in WSNs, the authors of [4] propose a congestion-based method for modelling and verifying WSNs using Place/Transition nets. Their proposed model only focus on the distribution of packets on a WSN. They have developed a tool called WSN-PN which allows for editing a network topology, generating automatically a corresponding PN model, and detecting congestion by verifying LTL formulae on the modelled PN. Moreover, WSN-PN supports users to abstract components, which can be either sensors or channels, applying to the modelled PN.

### 1.2 Outline

Section 2 recalls some basic concepts of CPNs. Section 3 proposes CPN-based model of WSNs. Section 4 shows analysing of the model to detect congestion in WSNs. More extensive experiments are also reported in this section. Finally, Section 5 draws conclusion and outlines future work.

## 2. PRELIMINARIES

Coloured Petri nets (CPNs for short) [9, 10, 20] is a modelling language which combines the strengths of Petri nets with the expressive power of functional programming languages. In CPNs, tokens are distinguished by the "colour" instead of only the "black" one. Moreover, arc expressions (an extended version of arc weights in classical Petri nets) specify which tokens can flow over the arcs. Guards that are Boolean expressions defining additional constraints on enabling transitions.

Let $EXP$ be the set of expressions complying with the CPN-ML [21] syntax. $EXP$ is used to express components of CPN such as arc expressions, guards. Before expressions are evaluated to values, the variables in the expressions must be assigned values, which is called *binding*. Besides, a multiset is a set in which there can be several occurrences of the same token. Multiset is defined in Definition 1 [20]. It is an important concept used in the later definitions of markings, steps, and the enabling and occurrence of transitions. Definition 2 [20] gives the formal definition of CPNs.

DEFINITION 1. Let $S = \{s_1, s_2, ...\}$ be a non-empty set. A *multiset* over $S$ is a function $m : S \to \mathbb{N}$ that maps each element $s \in S$ into a non-negative integer $m(s) \in \mathbb{N}$ called the *number of appearances* (coefficient) of $s$ in $m$. A multiset $m$ can also be written as a sum:

$$^{++}\sum_{s \in S} m(s)`s = m(s1)`s1 + +m(s2)`s2 + +...$$

*Membership*, *addition*, *comparison*, and *subtraction* are defined as follows, where $m_1$, $m_2$, and $m$ are multisets:

1. $\forall s \in S : s \in m \Leftrightarrow m(s) > 0$
2. $\forall s \in S : (m_1 + +m_2)(s) = m_1(s) + m_2(s)$
3. $m_1 <<= m_2 \Leftrightarrow \forall s \in S : m_1(s) \leq m_2(s)$
4. When $m_1 <<= m_2$,
$\forall s \in S : (m_2 - -m_1)(s) = m_2(s) - m_1(s)$

The set of all multisets over $S$ is denoted $S_{MS}$.

DEFINITION 2. A *Coloured Petri net* is a nine-tuple $\langle P, T, F, \Sigma, V, c, g, f, m_0 \rangle$ where:

1. $P$ is a finite, nonempty set of *places*.
2. $T$ is a finite, nonempty set of *transitions* such that $P \cap T = \emptyset$.
3. $F \subseteq P \times T \cup T \times P$ is a finite set of directed *arcs*.
4. $\Sigma$ is a finite set of non-empty *colour sets*.
5. $V$ is a finite set of *typed variables* such that $Type[v] \in \Sigma$ for all $v \in V$.
6. $c : P \rightarrow \Sigma$ is a *colour set function* that assigns to each place $p \in P$ a colour set $c(p) \in \Sigma$.
7. $g : T \rightarrow EXP$ is a *guard function* that assigns to each transition $t \in T$ a guard expression of the Boolean type.
8. $f : F \rightarrow EXP$ is an *arc expression function* that assigns to each arc $a \in F$ an arc expression of a multiset type $c(p)_{MS}$, where $p$ is the place connected to the arc $a$.
9. $m_0 : P \rightarrow EXP$ is an *initialisation function* that assigns to each place $p \in P$ an initialisation expression of a multiset type $c(p)_{MS}$.

DEFINITION 3. For a Coloured Petri net $CPN = \langle P, T, F, \Sigma, c, g, f, m_0 \rangle$, we define the following concepts:

1. A *marking* is a function $M$ that maps each place $p \in P$ into a multiset of tokens where $M(p) \in C(p)_{MS}$. $m_0$ is *initial marking* of $CPN$.
2. The *variables of a transition* $t$ are denoted $Var(t) \in V$ and consit of the free variables appearing in guard of $t$ and in the arc expressions of arcs connected to $t$.
3. A *binding* of a transition $t$ is a function $b$ that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. The set of all bindings for a transition $t$ is denoted $B(t)$.
4. A *binding element* is a pair $(t, b)$ such that $t \in T$ and $b \in B(t)$. The set of all binding elements $BE(t)$ for a transition $t$ is defined by $BE(t) = \{(t, b) | b \in B(t)\}$. The set of all binding elements in a CPN model is denoted $BE$.
5. A *step* $Y \in BE_{MS}$ is a non-empty, finite multiset of binding elements.

Definition 3 [20] defines concepts used to express the semantics of CPNs. Then the enabling and occurrence of a binding element are summarised in Definition 4 [20].

DEFINITION 4. A binding element $(t, b) \in BE$ is *enabled* in a marking $M$ if and only if the following two properties are satisfied:

1. $G(t)\langle b \rangle = true$
2. $\forall p \in P : E(p, t)\langle b \rangle <<= M(p)$

When $(t, b)$ is *enabled* in $M$, it may occur, leading to the marking $M'$ defined by:

3. $\forall p \in P : M'(p) = (M(p) - -E(p, t)\langle b \rangle) ++ E(t, p)\langle b \rangle$

$M'$ is directly reachable from $M$. This is denoted by $M \xrightarrow{(t,b)} M'$. The set of markings reachable from a marking $M$ is denoted $\mathfrak{R}(M)$.

A CPN model can be organised as a set of hierarchically related modules. CPN models with modules are also called *hierarchical Coloured Petri Nets* (HCPNs for short) [9, 20]. A HCPN allow dividing a module into smaller modules (sub modules) connected to each other using well-defined interfaces (substitution transitions and fusion places). With HCPNs, human modeller can work at different abstraction levels and concentrate on only a few details at a time. Moreover, a module is defined once and is used repeatedly. This allows reading only one description, and to modify one description when changes are necessary.

## 3. MODELLING WIRELESS SENSOR NETWORKS USING COLOURED PETRI NETS

In this section, we assume that the reader is familiarised with the Coloured Petri nets concept.

In this paper, we use a WSN whose topology depicted in Figure 1, as a straight case study.

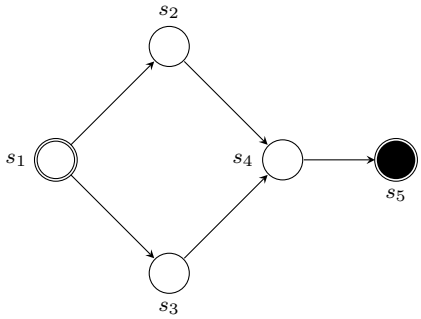



Fig. 1: Topology of example WSN. A double-lined circle denotes a source sensor, a full circle denotes a sink sensor, and a single circle denotes an intermediate sensor.

The WSN includes 5 sensors and 5 channels be long to four types of components (source, intermediate, sink, channel). The configuration of the WSN is shown in Table 1. Herein, we configure same parameters for components which are same type. In each sensor, we have a pair of buffer and queue. The buffer contains the packets a sensor is receiving. Then, the buffered packets are sequentially transferred to the queue, in order to be sent by the sensor. The processing rate indicates the number of packets a sensor can handle (transfer from buffer to queue) over a given period, while the sending rate specifies the number of packets sent by a sensor to its connected channels. In turn, transmission rate specifies the number of packets which are transfer from channel to sensor. Total packets stands for all packets which sources generated.

Table 1. : The parameter configuration of the WSN in Figure 1.

| | **Source** | **Sink** | **Intermediate** | **Channel** |
|---|---|---|---|---|
| **Sending rate** | 3 | 3 | 3 | N/A |
| **Buffer size** | 5 | 5 | 5 | 5 |
| **Queue size** | 5 | 5 | 5 | N/A |
| **Processing rate** | 5 | 5 | 5 | N/A |
| **Transmission rate** | N/A | N/A | N/A | 3 |
| **Total packets** | 6 | N/A | N/A | N/A |

### 3.1 CPN Model for WSN

Based on characteristics of a WSN, there are two different generic components: sensor nodes and channels. Each of these components has some behaviours that can be modelled by CPNs.

Sensor nodes are divided into three types including source sensor, intermediate sensor, and sink sensor. The behaviours of each type are as follows:

—A source sensor generates packets and sends these packets into its next hops.

—A intermediate and a sink sensor process packets internally, i.e. they move packets from their buffers to their queues.

A channel component includes following behaviours:

—receive packets sent from in-connected sensors

—send (transmit) packets to out-connected sensors

According to the previous presentation, the CPN-based model of a WSN is composed of seven modules. Figure 2 shows module hierarchy of the proposed model. The "Top" module is prime module with no incoming arc. It is decomposed into two sub-modules "Initialisation" and "Processing". The "Initialisation" module is responsible to initialise markings for working places of the overall model. The "Processing" module includes four sub-modules such as "Generate Packet", "Internal Process", "Receive Packet", and "Transmit Packet" corresponding to main operations of all WSN components.

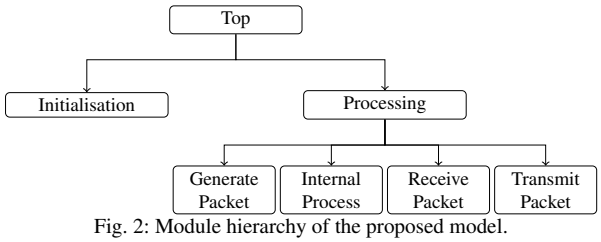


Fig. 2: Module hierarchy of the proposed model.

Figure 3 demonstrates the most abstract "Top" module, which includes two substitution transitions "initialise" and "process" (associated with the "Initialisation" module and the "Processing" module respectively) with initial information place "network" and two working places "sensors" and "channels". The declarations of the colour sets used in this module are listed in Listing 1. The declarations are presented using the CPN-ML syntax [21]. We divide such colour sets into two types: (a) colour sets describe parameters of net components and structure of the topology of the WSN and (b) colour sets describe parameters of overall WSN.

Colour set type (a) is specified as follows:

—Colour set "SENSOR_TYPE" is an enumeration colour set with three values "Source", "Intermediate", and "Sink" corresponding to three types of sensors.

—Colour set "SENSOR_PARAM" includes eight fields describing configuration and state of a sensor. Such fields are "typ" (types of sensors), "buf" (current packets in buffer), "queue" (current packets in queue), "pmax" (current maximum packets that a source sensor can generate – this field is only mean of source sensors), "sending_rate" (sending rate of sensor), "processing_rate" (processing rate of sensor), "buf_size" (size of buffer), and "queue_size" (size of queue).

—Colour set "CHANNEL_PARAM" includes three fields describing configuration and state of a channel such as "channel_buf" (current packets in channel buffer), "trans_rate" (transmission rate), and "channel_buf_size" (size of channel buffer).

—Colour set "SENSOR" is defined as a product of simple colour set "INT" (representing id of a sensor) and "SENSOR_PARAM" (representing parameters of the sensor).

—Colour set "CHANNEL" stands for a channel and it is defined as a product of three colour sets as follows: id of from-sensor of the channel ("INT"), list of remaining information of the channel ("CHANNEL_TO_PARTS"), and id of the channel ("INT"). Colour set "CHANNEL_TO_PARTS" is defined as a list of a product of two colour sets "CHANNEL_PARAM" and "INT". The reason of defining remaining information of the channel as a list is explained in next sections. Herein, colour set "CHANNEL_PARAM" stands for parameters of the channel while colour set "INT" stands for id of to-sensor of the channel. For example, channel $s_1 - s_2$ can be expressed by $channel$ value defined as follows:

$$Type[channel] = \text{CHANNEL};$$
$$channel = (1, [(\{channel\_buf = 0,$$
$$trans\_rate = 3, channel\_buf\_size = 5\}, 2)], 1)$$

Colour set type (b) is specified as follows:

—Colour set "CHANNEL_MODE" is an enumeration colour set representing channel modes supported in the proposed model including "Unicast" an "Broadcast". This paper does not consider "Multicast" mode.

—Colour set "NETWORK" includes four fields representing parameters of overall network as follows: "channel_mode", net_sensors (list of sensors in the network), and net_channels (list of channels in the network).

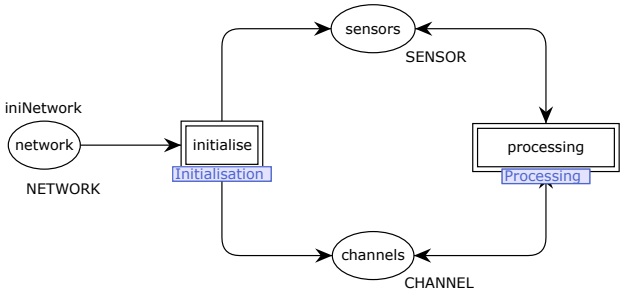


Fig. 3: The module "Top".

Listing 1: Declarations of the colour sets.

```
colset SENSOR_TYPE = with Source | Intermediate | Sink;

colset SENSOR_PARAM = record typ:SENSOR_TYPE * buf:INT
  * queue:INT * pmax:INT * sending_rate:INT
  * processing_rate:INT * buf_size:INT * queue_size:INT;
colset CHANNEL_PARAM = record channel_buf:INT
  * trans_rate:INT * channel_buf_size:INT;

colset SENSOR = product INT * SENSOR_PARAM;
colset CHANNEL_TO_PARTS = list (product CHANNEL_PARAM
  * INT)
colset CHANNEL = product INT * CHANNEL_TO_PARTS * INT;

colset CHANNEL_MODE = with Unicast | Broadcast;
colset NET_SENSORS = list SENSOR;
colset NET_CHANNELS = list CHANNEL;

colset NETWORK = record channel_mode:CHANNEL_MODE
  * net_sensors:NET_SENSORS * net_channels:NET_CHANNELS;
```

Listing 2 expresses value of constant "iniNetwork" assigned to initial marking of place "network" of the "Top" module. This value is set based on the configuration in Table 1. A user can easily change the value per his requirement.

Listing 2: Initial marking of the "network" place.

```
val iniSensors = [(1,{typ = Source,buf = 0,queue = 0,
  pmax = 6,sending_rate = 3,processing_rate = 5,
  buf_size = 5,queue_size = 5}),(2,{typ =
  Intermediate,buf = 0,queue = 0,pmax = 0,
  sending_rate = 3,processing_rate = 5,buf_size = 5,
  queue_size = 5}),(3,{typ = Intermediate,buf = 0,
  queue = 0,pmax = 0,sending_rate = 3,
  processing_rate = 5,buf_size = 5,queue_size = 5})
  ,(4,{typ = Intermediate,buf = 0,queue = 0, pmax =
  0,sending_rate = 3,processing_rate = 5,buf_size =
  5,queue_size = 5}),(5,{typ = Sink,buf = 0,queue =
  0,pmax = 0,sending_rate = 3,processing_rate = 5,
  buf_size = 5,queue_size = 5})];

val iniChannels = [(1,[({channel_buf = 0,trans_rate =
```

```
  3,channel_buf_size = 5},2)],1),(1,[({channel_buf =
  0,trans_rate = 3,channel_buf_size = 5},3)],2)
  ,(2,[({channel_buf = 0,trans_rate = 3,
  channel_buf_size = 5},4)],3),(3,[({channel_buf =
  0,trans_rate = 3,channel_buf_size = 5},4)],4)
  ,(4,[({channel_buf = 0,trans_rate = 3,
  channel_buf_size = 5},5)],5)];

val iniChannelMode = Broadcast;

val iniNetwork = {channel_mode = iniChannelMode,
  net_sensors = iniSensors,net_channels = iniChannels}
```

*The "Initialisation" module.* This module intends to distinguish and convert information of place "network" to two working places "sensors"and "channels". For example, with the initial marking of place "network" depicted in Listing 2, when this module is executed, the new markings of place "sensors" will be as follows: $M(sensors) = list\_to\_ms(iniSensors)$[2]. The new marking of place "channels" has a different of place "sensors", its value depends on channel mode of the WSN. In unicast mode, simply $M_{unicast}(channels) = list\_to\_ms(iniChannels)$. In broadcast mode, all channels, which have identical from-sensor id, are combined into a channel; for instant,

$$M_{broadcast}(channels) =$$
$$(1,[(\{channel\_buf = 0, trans\_rate = 3,$$
$$channel\_buf\_size = 5\},2),(\{channel\_buf = 0,$$
$$trans\_rate = 3, channel\_buf\_size = 5\},3)],1)$$
$$++(2,(\{...\},4),2)++(3,(\{...\},4),3)++(4,(\{...\},5),4)$$

This can be performed because the "CHANNEL_TO_PARTS" part of colour set "CHANNEL" is defined as a list.

*The "Processing" module.* This module represents the main operations of the WSN. It is the key factor of overall model. Figure 4 demonstrates the "Processing" module, which is decomposed into four sub-modules including "Generate Packet", "Internal Process", "Receive Packet", and "Transmit Packet" which are associated with four substitution transitions "generate", "process", "receive", and "transmit" respectively.

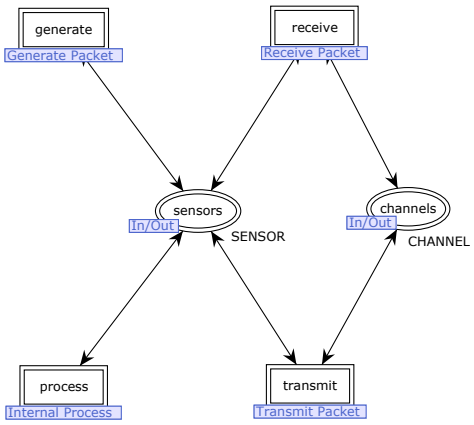



Fig. 4: The "Processing" module.

Figure 5 depicts the "Generate Packet" module which corresponds to the generating packets operation of a source sensor (identified by $s\_id$). Function "guard_generate" defines a guard which expresses firing condition of transition "generate packet". This transition can fire if number of generable packets of the sensor is greater than 0 and number of current packets in queue of the sensor is less than the queue size. Let $n\_generated$ be number of generated packets which is defined as minimum of current number of generable packets ($\#pmax\ s\_param$) of the sensor, sending rate of the sensor ($\#sending\_rate\ s\_param$), and subtraction of queue size and $\#pmax\ \ s\_param$. The specification assurances that queue of the sensor is never overloaded. Function "generate" updates state of the sensor. First, it subs $n\_generated$ from current number of generable packets. Then, it adds $n\_generated$ to queue of the sensor. All rest of information of the sensor are unchanged.

[2] $list\_to\_ms$ is a function converting a list to a multiset. For example, $list\_to\_ms([1,2]) = 1`1 + +1`2$

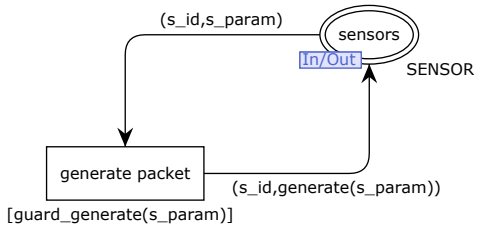



Fig. 5: The "Generate Packet" module.

The "Internal Process" module corresponds to internal processing operation of an intermediate or sink sensor, as shown in Figure 6. The behaviour of this operation is to move packets from buffer of the sensor into its queue. Number of moving packets which is determined as minimum of number of current packets in sensor buffer, processing rate of the sensor, and subtraction of queue size of the sensor and number of current packets in the queue. Function "guard_process" defines a guard expressing firing condition of transition "internal process". This function returns true value if the desired sensor is intermediate or sink sensor (is not source sensor), number of current packets in sensor buffer is greater 0, and number of current packets in sensor queue is less than queue size.

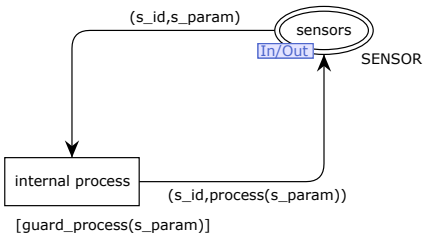



Fig. 6: The "Internal Process" module.

Figure 7 depicts the "Receive Packet" module which corresponds to the operation that a sensor sends packets and its connected channels receive these packets. The selection of channels receiving packets is based on channel mode (unicast, broadcast) of the WSN. The selection is expressed in the "Initialisation" module before. Function "guard_receive" is assigned to guard of transition "receive packet". This transition can fire if number of current packets in queue of the from sensor (identified by $from\_s\_id$) is greater than 0. This operation only depends on state of the from sensor, it does not depend on state of these connected channels. Let $n\_received$ be number of sent/received packets which is defined as minimum of number of current packets in sensor queue and sending rate of the sensor. Function "send" updates state of the from sensor when the operation has done (i.e. transition "receive packet" has occurred). Queue of the sensor is subbed a mount $n\_received$. Function "receive" updates state of the selected channels (listed in variable $c\_list$). Each of these selected channels is added amount $n\_received$ to its buffer.

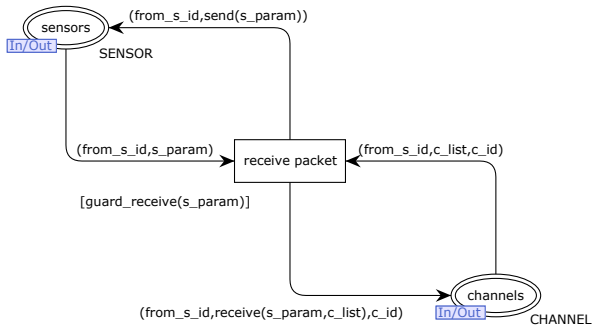



Fig. 7: The "Receive Packet" module.

Figure 8 demonstrates the "Transmit Packet" module which is responsible to the transmitting packets to a sensor of a channel. The operation depends on state of the channel, but not on channel mode or state of to sensor. Function "guard_transmit" is assigned to a guard expressing firing condition of transition "transmit packet". It returns true value if number of current packets in buffer of the channel is greater than 0. As shown in Figure 8, variable $from\_s\_id$ stands for the id of from sensor while $to\_s\_id$ stands for the id of to sensor of the channel. Function "transmit_from" updates state of the channel having to sensor id which equals $to\_s\_id$ in list of channels $c\_list$. Let $n\_transmitted$ be number of transmitting packets defined as minimum of number of current packets in channel buffer and transmission rate of the channel. The buffer of this channel is subbed $n\_transmitted$ packets. The rest information of the channel are unchanged. Function "transmit_to" updates state of to sensor $to\_s\_id$. The buffer of the sensor is added $n\_transmitted$ packets. The rest information of the sensor are unchanged.

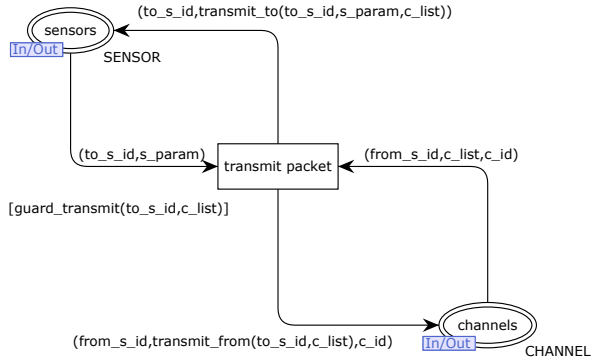



Fig. 8: The "Transmit Packet" module.

### 3.2 CPN Model with Concurrent Semantic

In a WSN, the sensor node executes a number of concurrent operations: generating packets (capturing information), processing, and sending packets; while the channel concurrently receives and transmits packets [14]. Moreover, a component can operate independently with each others. Therefore, operations of sensors and channels in a WSN should be performed concurrently. This is called concurrency of WSNs.

As discussed in Section 1, the CPN-based model with concurrent semantic is needed for describing WSN in reality. Since a binding element stands for an operation of a net component, some enabled binding elements of the CPN model need to occur simultaneously. Theoretically, a CPN model expresses concurrency by defining a new semantic called concurrent semantic which allows occur some enabled binding elements simultaneously [9]. But in implemental aspects, CPN Tools and another tools did not implement the concurrent semantic. Instead, they implemented sequential semantic (i.e. only one of enabled binding elements can occur in a marking and the selection of these binding elements is non-deterministic) with argument that a marking reached in concurrent semantic will be the same as the one which will be reached in sequential semantic. However, state space of a model with sequential semantic has too many intermediate markings against state space of this model with concurrent semantic. This claim reflects appropriately the observation of WSNs in [14]. Moreover, in congestion detection problem, using sequential semantic can lead delusory result which is not match with real WSN (see Section 4).

We propose a concurrent semantic which expresses the concurrency of WSNs. The formal definition of the concurrent semantic is presented below.

Let $EBE = \{ebe_i\}$ $(i = \overline{1,n})$ be set of enabled binding elements in marking $M$ of the sequential model. Each enabled binding element stands for an enabled operation of a component of the WSN. Since components of the WSN can operate concurrently, some enabled binding elements of $EBE$ can occur simultaneously. However, in unicast mode, a sensor must choose only on of its connected channels to send packets. Correspondingly, enabled binding elements of transition "receive packet" of the "Receive Packet" module which have the same from sensor (variable $s\_id$) can not occur concurrently. Therefore, we define a set called *unicast-set* which reflexes the constraint of unicast mode. unicast-set is defined in Definition 5.

DEFINITION 5. An *unicast-set* $U$ is a set of all enabled binding elements $\{(t_1, b_1), ..., (t_m, b_m)\}; m \geq 1$ such that:

1. $U \subseteq EBE$;
2. $t_i =$ "receive packet" with $\forall i \in \{1, 2, .., m\}$;
3. $b_i(from\_s\_id) = b_j(from\_s\_id)$ with $\forall i, j \in \{1, 2, .., m\}$;
4. and $U$ is a mutually exclusive set, i.e. the occurring of any subset of binding elements $U' \subset U$ results in a marking in which some other binding element $(t_j, b_j) \notin U'$ is disabled.

We divide $EBE$ into two groups. The first group includes all disjoint unicast-sets $U_1$,...,$U_m$ $(m \geq 0)$ of $EBE$, i.e. $U_i \cap U_j = \emptyset$ with $\forall i, j \in \{1, ..., m\}; i \neq j$. The second group is a set called $NC$ including all remaining enabled binding elements in $EBE$, i.e. $NC = EBE \backslash (\bigcup_{i=1}^{m} U_i)$. $NC$ is not in conflict and its enabled binding elements can occur simultaneously. Then we define a *w-step* (distinguish with the *step* concept in CPNs theory) as shown in Definition 6.

DEFINITION 6. A *w-step* $W$ of a $EBE$ is a set of enabled binding elements such that:

$W = NC \bigcup (\bigcup_{i=1}^{m} (t_i, b_i))$ with $(t_i, b_i) \in U_i$. A set of all possible w-steps of a $EBE$ is denoted by $WS$. The cardinality of $WS$ is specified as follows:

$$|WS| = \begin{cases} 1 & if\, m = 0 \\ \prod_{i=1}^{m} |U_i| & if\, m \geq 1 \end{cases}$$

The concurrent semantic is based on Theorem 7 and Theorem 8.

THEOREM 7. Let $Y$ be a w-step in marking $M$, then $Y$ is a non-conflict set.

PROOF. Let $(t_i, b_i)$ and $(t_j, b_j)$ be two arbitrary distinct enabled binding elements of $Y$. Assume $(t_i, b_i)$ occurs before $(t_j, b_j)$ reaching marking $M_1$. In the sequential model, when a binding element occurs, tokens in each place is unchanged, it

is only change inside values of tokens. Therefore, $(t_j, b_j)$ still satisfies resource constraints.
Let $to$ and $to_1$ are tokens in place "sensors" in marking $M$ and $M_1$ respectively, such that $to \xrightarrow{(t_i,b_i)} to_1$. Regarding guard conditions, if $(t_j, b_j)$ does not bind to $to_1$ then $(t_j, b_j)$ is still enabled, otherwise there are all possible cases:

—$t_i$ = "generate packet" or $t_i$ = "internal process" and $t_j$ = "receive packet". Because $(t_j, b_j)$ is enabled, then $G(t_j)\langle b_j\rangle_M = true$ i.e. $(\#q\ to > 0) = true$. Since $(\#q\ to_1) = (\#q\ to) + inc$ with $inc > 0$, we have $(\#q\ to' > 0) = true$ i.e. $G(t_j)\langle b_j\rangle_{M_1} = true$.

—$t_i$ = "receive packet" and $t_j$ = "transmit packet". We have, $G(t_j)\langle b_j\rangle_M = true$ i.e. $(\#cb\ to > 0) = true$. $(\#cb\ to_1) = (\#cb\ to) + inc$ with $inc > 0$, hence $(\#cb\ to_1 > 0) = true$ i.e. $G(t_j)\langle b_j\rangle_{M_1} = true$.

—$t_i$ = "transmit packet" and $t_j$ = "internal process". We have, $G(t_j)\langle b_j\rangle_M = true$ i.e. $(\#b\ to > 0) = true$. Sine $(\#b\ to_1) = (\#b\ to) + inc$ with $inc > 0$, we have $(\#b\ to_1 > 0) = true$ i.e. $G(t_j)\langle b_j\rangle_{M_1} = true$.

All cases indicate that the occurring of $(t_i, b_i)$ does not effect to enabling of $(t_j, b_j)$. Then, we can conclude that $Y$ is a non-conflict set. □

THEOREM 8. Let $Y$ be a w-step in marking M. The binding elements of $Y$ can occur sequentially (one by one) in any order and the marking reached will be the same as the one.

PROOF. Let $or = ((t_1, b_1), ...., (t_n, b_n))$ with $n \geq 1$ be an arbitrary order of all binding elements of $Y$ and $M'$ is marking reached by making the binding elements of $Y$ occur sequentially in this order. Let $M_1, ..., M_{n-1}$ be intermediate markings in occurrence sequence from $M$ to $M'$. Let $M \bigotimes (t_1, b_1)$ be effect of $(t_1, b_1)$ on $M$, we have

$$M_1 = M \bigotimes (t_1, b_1)$$
$$M_2 = M_1 \bigotimes (t_2, b_2)$$
$$...$$
$$M_{n-1} = M_{n-2} \bigotimes (t_{n-1}, b_{n-1})$$
$$M' = M_{n-1} \bigotimes (t_n, b_n)$$

By performing a series of substitutions, we obtain

$$M' = M \bigotimes (t_1, b_1) \bigotimes ... \bigotimes (t_n, b_n) (^*)$$

We prove that $M \bigotimes \gamma_i \bigotimes \gamma_j = M \bigotimes \gamma_j \bigotimes \gamma_i; \forall i, j \in \{1, ..., n\}$ and $i \neq j$ $(^{**})$. Effect of $\gamma_i$ is addition/subtraction of $inc$ packets $(inc > 0)$ to/from of a part $(b, q, cb)$ of a token in $M$. Subtraction can be replaced by addition with negation of the subtrahend, e.g. $3 - 1 = 3 + (-1)$. Addition is commutative. Therefore, we can derive $(^{**})$.
From $(^*)$ and $(^{**})$, we can conclude that $or$ has no influence on the total effect of all binding elements of $Y$, i.e. $M'$ is the same as the one in any order. □

We have, when a step $Y$ is enabled in a marking $M$, the binding elements of $Y$ can occur one by one in any order reaching the same marking $M'$. Moreover, we can determine a fixed order $or^*$ of the binding elements of $Y$ and control these occur sequentially in this order. That helps to reduce significantly the state space. Instead of having $|Y|!$ paths from $M$ to $M'$ in the state space we only have one path in them.

The new model called concurrent model is combination of the CPN-based model (see Section 3.1) and the concurrent semantic. In practical, since CPN Tools is used we express the concurrent model by a behaviour-equivalent model in CPN Tools. The behaviour-equivalent model is possible be constructed and analysed by CPN Tools.

## 4. CONGESTION DETECTION

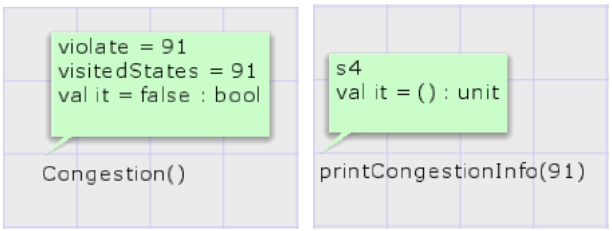
### 4.1 Finding Congestion

To easily find congestion, we adjust the sequential (concurrent) model. We add a transition called "congestion" into each model. This transition gets list of sensors and channels from places "sensor'" and "channels'" respectively. Then it processes the obtained information to get list of congestion sensors and channels, whose emptiness corresponds to congestion-free of the WSN. A sensor (channel) is congestion iff its sensor (channel) buffer is overload, i.e. number of current packets in the buffer is greater than the buffer size. In this research, we only focus on specifying a WSN whether occurs congestion. Therefore, when a congestion is found, overall model will be inactive. This assumption is implemented easily by configuring the model such that when transition "congestion" occurs, there are no enabled binding elements in the new marking i.e. this is a dead marking. Herein, the congestion detection problem is reduced to a reachability problem.

Some CPN-ML [21] code is used to check the reachability of the "congestion" transition on each model. The verification is performed based on ASK-CTL tool [22], a CTL model checker built in CPN Tools.

Figure 9 (a) shows result of the finding congestion of the concurrent model of the WSN in Figure 1 with broadcast mode is desired. The obtained result includes a boolean value which indicate whether occurs congestion ($it = false$ indicates occurring congestion while $it = true$ indicates is no occurring congestion), the identifier of violate arc ($violate = 91$), and the number of visited states ($visitedStates = 91$). In this context, a visited state is construed be a visited arc.

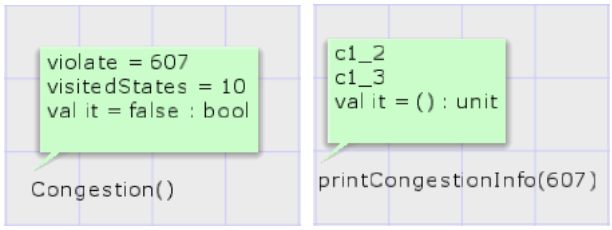
From the raw result of congestion detection, we continue to use built-in functions of CPN Tools to get more insight information. Figure 9 (b) represents the result, which shows that the congestion occurs in Sensor $s_4$.




(a) (b)

Fig. 9: The result of congestion detection of the concurrent model. (a) The raw result. (b) The more insight result.

Respectively, Figure 10 (a) shows result of the finding congestion of the sequential model of the WSN in Figure 1 with broadcast mode is desired. Figure 10 (b) indicates that the congestion occurs in Channel $s_1 - s_2$ and Channel $s_1 - s_3$.
Consider two congestion detection results of the sequential model and the concurrent model. The congestion result in the sequential model is delusory. The reason is that the state space of the sequential model has many intermediate markings which do not express real states of the WSN (see Figure 11). In while, the congestion result in the concurrent model is match with real behaviour of the WSN (see Figure 12).




(a) (b)

Fig. 10: The result of congestion detection of the sequential model. (a) The raw result. (b) The more insight result.

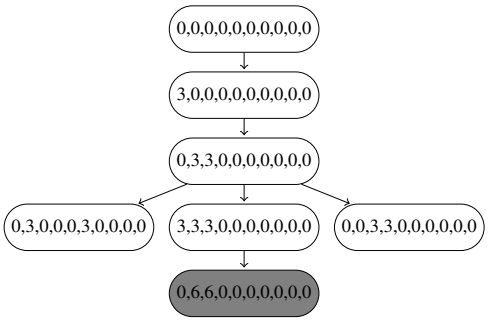



Fig. 11: Partial and condensed state space of the sequential model. A marking (node) is stood by a rounded rectangle. The inside content of a node includes: queue of Sensor $s_1$, buffer of Channel $s_1 - s_2$, buffer of Channel $s_1 - s_3$, buffer of Sensor $s_2$, queue of Sensor $s_2$, buffer of Sensor $s_3$, queue of Sensor $s_3$, buffer of Channel $s_2 - s_4$, buffer of Channel $s_3 - s_4$, buffer of Sensor $s_4$. The node, which is filled by grey colour, denotes a state occurring congestion.

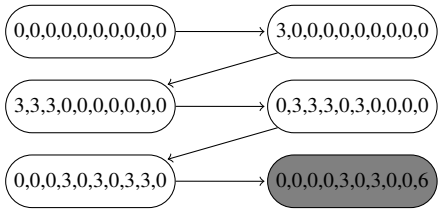



Fig. 12: Partial and condensed state space of the concurrent model. A marking (node) is stood by a rounded rectangle. The inside content of a node is specified same in Figure 11. The node, which is filled by grey colour, denotes a state occurring congestion.

## 4.2 Generating Counter Example

Counter example is a basic concept in model checking. It releases some information to conduct error diagnostics. In this context, we construct the shortest counter example – i.e. to find one of the shortest occurrence sequences leading from the initial marking to a marking containing an output arc where a predicate does not hold. The predicate is defined as no occurring of the "congestion" transition.

Some built-in functions of CPN Tools working on the state space is used to generate counter example. Figure 13 shows a counter example on the concurrent model (see Section 4.1) when detected congestion. Each line of this result represents list of information of operations which actually active in each concurrent updating phase. Information of an operation includes transition name and real component of the WSN conducting this operation. For example, $gen(s1)$ denotes that Sensor $s_1$ conducts generating packets operation, and $trans(s1 - s2)$ denotes that Channel $s_1 - s_2$ conducts transmitting packets operation to Sensor $s_2$.

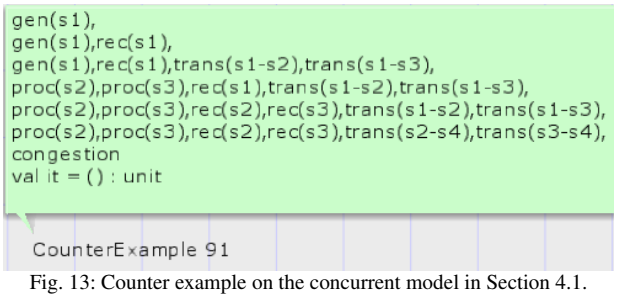



Fig. 13: Counter example on the concurrent model in Section 4.1.

## 4.3 Experimental Results

Table 2 shows experimental results of congestion detection on concurrent models of WSNs. The WSNs are considered in broadcast mode. Numbers of sensors range from 1 to 100.

From the experimental results, there are some observations of WSN as follows. Fist, the number of nodes of the state space equals the number of arcs. It is properly since the concurrent model is deterministic in broadcast mode. Second, increasing of packets (increasing of data rates) makes the congestion occurs early (decreasing of visited states). This is consistent with the results presented in [3].

At the moments, the number of sensors in a WSN is still limited at 100, but once combined with appropriate state space reduction techniques as suggested in Section 5, this number can be increased significantly. This opens an interesting direction for our future work.

Table 2. : Experimental results of congestion detection on concurrent models of WSNs. Columns $sensor$ and $packets$ denote number of sensors and packets of a WSN respectively; columns $nodes$ and $arcs$ denote number of nodes and arcs of the state space of the WSN; columns $result$ denotes congestion detection result ($false$ indicates occurring congestion while $true$ indicates is no occurring congestion); and $visitedarcs$ denotes number of visited arcs in verification.

| **Network** | | **State Space** | | **Congestion Detection** | |
|---|---|---|---|---|---|
| sensors | packets | nodes | arcs | result | visited arcs |
| 10 | 10 | 793 | 793 | true | 793 |
| 10 | 20 | 892 | 892 | true | 892 |
| 10 | 40 | 1189 | 1189 | true | 1189 |
| 15 | 10 | 1697 | 1697 | true | 1697 |
| 15 | 20 | 2545 | 2545 | true | 2545 |
| 15 | 40 | 4294 | 4294 | false | 1008 |
| 30 | 10 | 3396 | 3396 | true | 3396 |
| 30 | 20 | 3493 | 3493 | true | 3493 |
| 30 | 40 | 5433 | 5433 | true | 5433 |
| 40 | 10 | 6121 | 6121 | true | 6121 |
| 40 | 20 | 8705 | 8705 | true | 8705 |
| 40 | 40 | 15233 | 15233 | false | 3809 |
| 50 | 10 | 9170 | 9170 | true | 9170 |
| 50 | 20 | 14879 | 14879 | false | 7267 |
| 50 | 40 | 26989 | 26989 | false | 4845 |
| 60 | 10 | 22792 | 22792 | false | 5113 |
| 60 | 20 | 42175 | 42175 | false | 4048 |
| 60 | 40 | 80941 | 80941 | false | 3622 |
| 80 | 10 | 33783 | 33783 | false | 9577 |
| 80 | 20 | 57457 | 57457 | false | 6651 |
| 80 | 40 | 105869 | 105869 | false | 5055 |
| 100 | 10 | 119022 | 119022 | false | 12493 |
| 100 | 20 | 226245 | 226245 | false | 8676 |
| 100 | 40 | 358799 | 358799 | false | 6594 |

## 5. CONCLUSION

This paper has presented a CPN-based model modelling the behaviours of a WSN. The proposed model represents the operations of each WSN component: sensors and channels. The modelled operations includes: generating packets of source sensors, internal processing of intermediate and sink sensors, sending/receiving packets of sensors/channels, and transmitting packets of channels.

Operations of WSN sensors and channels can operate concurrently. Consequently, some enabled binding elements in a marking of the sequential model can occur simultaneously instead only one of these enabled binding elements can occur non-deterministically. A new model, called concurrent model, has been proposed to express the concurrency of WSNs. The concurrent model is the combination of the sequential model and concurrent semantics.

The congestion detection problem has been reduced to a reachability problem. The verification is performed by using ASK-CTL tool working on the generated state space of the model. Counter example also has been generated to illustrate the path from the initial state to the congestion point which helps to expose root cause of the congestion.

The experiment still limits number of sensor at 100 because of the state space explosion problem. We intend to overcome this problem by using applying some state space reduction techniques such as abstraction, unfolding, sweep-line.

In practice, each component of a WSN exists an unreliability factor which indicates probability for that such component do not operate its functionalities correctly. Consequently, an enabled binding element in the CPN model of the WSN may probabilistically occur or not occur. A probabilistic model is naturally needed in this context. In the future, we intend to construct a probabilistic model for WSN, then to define and detect congestion with probabilities. We also will extend the proposed approach to time Petri net models, so as modelling the different timings in WSNs such as a randomised time for sending packets or a processing time by the sensor. Combination of probabilistic models and timed models is a promising work.

### Acknowledgment

This research is funded by Ho Chi Minh City University of Technology under grant number T-KHKT-2015-31.

## 6. REFERENCES

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[3] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell. CODA: congestion detection and avoidance in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 266–279. ACM, 2003.

[4] Khanh Le, Thang Bui, Tho Quan, Laure Petrucci, and Etienne Andre. Congestion verification on abstracted wireless sensor networks with the WSN-PN tool. *Journal of Advances in Computer Networks*, 4(1), 2016.

[5] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147. ACM, 2004.

[6] Charalambos Sergiou, Pavlos Antoniou, and Vasos Vassiliou. A comprehensive survey of congestion control protocols in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 16(4):1839–1859, 2014.

[7] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[8] András Varga and Rudolf 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, Marseille, France, March 3-7, 2008*, page 60, 2008.

[9] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and CPN Tools for modelling and validation of concurrent systems. *STTT*, 9(3-4):213–254, 2007.

[10] Fei Liu and Ming Yang. Compositional colored petri net approach to multiscale modeling for systems biology. *International Journal of Modeling, Simulation, and Scientific Computing*, 5(04):1450017, 2014.

[11] Bruno Lacerda and Pedro U Lima. Petri nets as an analysis tool for data flow in wireless sensor networks. In *The First Portuguese Conference on WSNs, Coimbra, Portugal*, pages 1–6, 2011.

[12] Sajeh Zairi, NIEL Eric, and Belhassen ZOUARI. Global generic model for formal validation of the wireless sensor networks properties. *IFAC Proceedings Volumes*, 44(1):5395–5400, 2011.

[13] Dina M Ibrahim, Elsayed A Sallam, Tarek E Eltobely, and Mahmoud M Fahmy. Coloured petri net model for vector-based forwarding routing protocol. In *The International Conference on Computing Technology and Information Management (ICCTIM)*, pages 169–176. Society of Digital Information and Wireless Communication, 2014.

[14] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGOPS operating systems review*, 34(5):93–104, 2000.

[15] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[16] D. A. Zaitsev. Switched LAN simulation by colored Petri nets. *Mathematics and Computers in Simulation*, 65(3):245–249, 2004.

[17] Congzhe Zhang and Mengchu Zhou. A stochastic Petri net-approach to modeling and analysis of *ad hoc* network. In *Information Technology: Research and Education (ITRE 2003)*, pages 152–156. IEEE, 2003.

[18] Mohammad Abdollahi Azgomi and Ali Khalili. Performance evaluation of sensor medium access control protocol using coloured petri nets. *Electronic Notes in Theoretical Computer Science*, 242(2):31–42, 2009.

[19] Antônio Dâmaso, Nelson Rosa, and Paulo Maciel. Using coloured petri nets for evaluating the power consumption of wireless sensor networks. *International Journal of Distributed Sensor Networks*, 2014, 2014.

[20] Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

[21] Soren Christensen and Torben Bisgaard Haagh. Design/CPN overview of CPN ML syntax. *University of Aarhus*, 3, 1996.

[22] Soren Christensen and Kjeld H Mortensen. Design/CPN ASK-CTL manual. *University of Aarhus*, 1996.