IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 11, 2022, accepted April 21, 2022, date of publication April 25, 2022, date of current version May 3, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3170094

Dynamic Slicing of Time Petri Net Based on MTL Property

P. CHARIYATHITIPONG AND W. VATANAWOOD
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330, Thailand
Corresponding author: W. Vatanawood (wiwat@chula.ac.th)

ABSTRACT The time Petri net (TPN) is a powerful tool for modeling, simulating, and verifying real-time systems. Unfortunately, the state spaces of the time Petri net grow exponentially due to the complexity of real-time systems. The enormous size of the state spaces could also cause state explosion problems in model checking. This paper proposes an alternative dynamic slicing algorithm written as a metric temporal logic (MTL) formula to reduce the size of the time Petri net model by considering specific criteria. Our algorithm proposes an alternative dependency graph representing the global firing time interval of the transition to remove the transitions that are never fired and do not affect the state space. The dynamic slicing algorithm involves the initial marking and the properties of the target metric temporal logic formula. Unlike the unsliced time Petri net, the result preserves all necessary execution paths for the model checking of a particular metric temporal logic formula. Therefore, model checking can generate sufficient state space to conduct verification equivalent to the unsliced time Petri net but may take less time, including if the unsliced time Petri net causes state space explosion.

INDEX TERMS Dynamic slicing, global firing dependency graph, metric temporal logic, real-time systems, slicing for verification, state space reduction, structural reduction, time Petri net.

I. INTRODUCTION

The time Petri net (TPN) is a mathematical model commonly used to represent and capture a real-time system's behavior, especially its concurrent, asynchronous, distributed, parallel, and nondeterministic operations. The TPN can implement all possible execution paths in a real-time system through recursive paths using the given initial marking and specific time constraints. The corresponding state space of the real-time system can be computed or generated using the TPN model to demonstrate the abstraction of its behavior. However, as the system becomes more complex, the size of its state space grows exponentially, which causes a state space explosion problem, a common obstacle in the model checking process. To address this situation, state space reductions based on various techniques are commonly exploited to reduce the number of redundant states visited during runtime state exploration. Moreover, the state space of a TPN is based on the markings of all its places and the time interval constraints of the firing sequences of its transitions, making it even more complex. Therefore, a state class graph (SCG) [1] is alternatively used to abstract the specific state space of a TPN, and an SCG is efficiently generated based on its reachability properties [2], [3].

Various studies have applied on-the-fly reduction to reduce the state space of a TPN by constructing a minimal state space on demand [4] and by limiting the state exploration in the on-the-fly mode based on the properties of timed computation tree logic (TCTL) [5], [6] or linear temporal logic (LTL) [7]. Partial order reduction has also been applied to construct the state space, but only in the same independent state as executing different firing sequences [8]–[11]. In [12], a stubborn set method for TPNs that concerns only the transitions and the firing constraints of the reachable firing rules was proposed. The stubborn sets are the transition sets that are not affected or reached by those outside the stubborn sets. A symmetry reduction method for TPNs to alleviate the combinatorial explosion problem by abstracting only one out of two equivalent states in the state class graph was also proposed in [13].

Furthermore, structural analysis of the formal model using pruning algorithms could also be conducted beforehand to further limit the unnecessary or unreachable structure of a TPN with regard to the target criterion.

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

Several TPN structural reduction methods have been proposed. For example, sets of heuristic reduction rules to reduce the size of a TPN were proposed in [14]–[16]. In [16] and [17], a TPN model was modularized into TPN subnets that preserve the original liveness and safety properties.

Model slicing is another structural reduction method used to minimize the size of an original model. Fortunately, the resulting TPN slice performs equivalently to the original TPN with respect to the target criteria in the verification process. This model slicing method is classified as either static or dynamic slicing. Static slicing considers only the slicing criteria while dynamic slicing considers the initial markings, the so-called initial state, and the target criteria. Model slicing is additionally used in software verification to reduce the formal model in the preprocessing phase [18].

In [19], a static slicing algorithm for slicing a formal model written in the Rebeca language based on the Rebeca dependency graph (RDG) was proposed, and the resulting slice preserved the deadlock property. Then, in [20], an algorithm considering the $\mathrm{LTL_x}$ property was constructed to slice the original model. A behavior tree dependence graph (BTDG) was used to construct slices for any target criterion related to the $\mathrm{LTL_x}$ property.

The asynchronous system model written in the Promela program was also sliced before being verified in [21] and [22]. In [23]–[25], slicing algorithms for timed automata were proposed. Several Petri net slicing algorithms to alleviate the state space explosion that occurs during model checking under limited conditions have been proposed. For example, two slicing algorithms were used to reduce a Petri net model that maintained the safety property in [26] and the $\mathrm{CTL^*_{-x}}$ property in [27]. Moreover, a place-invariant-based (PI-based) algorithm to slice Petri nets and preserve the place-invariant property was proposed in [28]. Furthermore, [29] proposed dynamic slicing algorithms for Petri nets by computing the backward and forward slices while considering an initial marking, and these algorithms were further improved using a structural dependency graph (SDG) in [30].

Nevertheless, the firing rules of Petri nets consider only the markings and ignore the firing time constraints while a TPN is more concerned with the relative firing time of each transition. This paper proposes an alternative structural analysis technique to slice the unnecessary structure of a model for a specific criterion. We intend to eliminate the irrelevant TPN elements to the criteria written in the metric temporal logic (MTL) properties and leave any other elements. Thus, the slicing algorithm preserves the target MTL properties of a TPN. We focus on the dynamic slicing approach in which both the initial marking and target criteria are considered and provide the traceability of the paths of fired tokens. We propose a new dependency graph called a firing dependency graph (FDG) representing all the dependencies within the global firing time interval of the transitions. First, we provide an algorithm to construct the firing dependency graph from the TPN model. Second, our dynamic slicing algorithm for a TPN to slice the TPN model based on the target criteria within the properties of the MTL is also proposed.

The remainder of this paper is organized as follows. Section II briefly introduces the time Petri net and the metric temporal logic. Section III illustrates the slicing TPN algorithm to reduce the TPN model based on these criteria. The case study is presented in Section IV, and Section V presents our conclusions.

II. PRELIMINARY

Verifying and analyzing real-time systems has two elements: the semantic models to describe the properties of a system and the specification of systems. This paper used the TPN model and MTL to represent the semantic model and the specification of the system, respectively. Therefore, this section introduces TPN and MTL concepts.

A. TIME PETRI NET

The time Petri net is a bipartite directed graph with two nodes: places and transitions. A circle depicts a place, and a bar represents a transition associated with the deterministic firing time. A directed arc connects place and transition. In addition, each circle may contain one or more dots, called tokens. Definition 1 formally defines the time Petri net.

Definition 1 (Time Petri net): A time Petri net is a 6-tuple $TPN = (P, T, B, F, m_0, st)$, where

- P is a finite nonempty set of places.
- T is a finite nonempty set of transitions, $P \cap T = \emptyset$.
- $B : (P \times T) \rightarrow \mathbb{N}$ is the backward incidence function or input function that defines the weight of each directed input arc from the places to transitions, where $\mathbb{N}$ is the set of nonnegative integers.
- $F : (T \times P) \rightarrow \mathbb{N}$ is the forward incidence function or output function that defines the weight of each directed output arc from the transitions to the places.
- m_0 is the initial marking that defines the tokens in each TPN place. Function $m_0(p)$ denotes the marking of place $p \in P$.
- $st : T \rightarrow \mathbb{Q}^* \times (\mathbb{Q}^* \cup \{\infty\})$ is the static firing interval function that defines the earliest and latest static firing times of each transition, where $\mathbb{Q}^*$ is the set of positive rational numbers. For a transition t, $st(t) = [est(t), lst(t)]$, where $est(t)$ and $lst(t)$ are the earliest and latest static firing times of transition t, respectively, satisfying $est(t) \leq lst(t)$.

The following standard notation denotes the input and output places. If $B(p, t) > 0$, then $\bullet t$ denotes the *input place set* of transition t. Similarly, $\bullet p$ represents the *input transition set* of place p.

Furthermore, if $F(t, p) > 0$, then $t\bullet$ denotes the *output place set* of transition t and $p\bullet$ represents the *output transition set* of place p.

Alternatively, the incidence matrices containing the connections from places to transitions and vice versa represent a particular TPN character. In practice, the input and output flow matrices construct the incidence matrices. The input flow matrix contains the input flows from places to

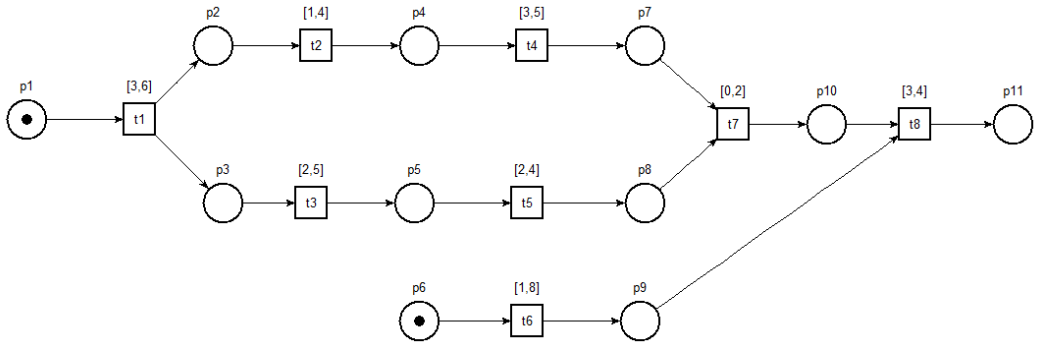

FIGURE 1. An example time petri net model.

transitions, and the output flow matrix contains the output flows from transitions to places.

For a *TPN* with n transitions and m places, the input flow matrix is an $m \times n$ matrix of integer $BM = [b_{ij}]$, where $b_{ij} = \begin{cases} 1, B(p_i, t_j) > 0 \\ 0, \text{otherwise} \end{cases}$. Similarly, the output flow matrix is an $n \times m$ matrix of integer $FM = [f_{ij}]$, where $f_{ij} = \begin{cases} 1, F(t_i, p_j) > 0 \\ 0, \text{otherwise} \end{cases}$.

B. METRIC TEMPORAL LOGIC

Metric temporal logic (MTL) is used to describe the properties of a real-time system. MTL formulae extend linear temporal logic (LTL) with adds time intervals bounds to the temporal operators.

Let P be a set of propositional variables. Then, the well-formed formulae of MTL are formed following abstract syntax:

- $\varphi := true|p$ where $p \in P$;
- $\varphi := \neg\varphi|\varphi_1 \wedge \varphi_2|\varphi_1 \vee \varphi_2|\varphi_1 \rightarrow \varphi_2$;
- $\varphi := \Box_{[t1,t2]}\varphi \left|\Diamond_{[t1,t2]}\varphi\right| \varphi_1 \mathcal{U}_{[t1,t2]}\varphi_2$ where $t1 \epsilon \mathbb{Q}^+$ and $t2 \epsilon \mathbb{Q}^+ \cup \{\infty\}$ with $t1 \leq t2$.

A time state sequence $\rho = (\pi, \tau)$ is a pair consisting of an infinite sequence π of states $\pi_i \subseteq P$ and a time point τ at state π_i such that $\pi_i \leq \pi_{i+1}$. The semantics of MTL is defined as follows.

- $(\rho, i) \models p$ iff $p \in \pi_i$;
- $(\rho, i) \models \neg\varphi$ iff $(\rho, i)\, \varphi$;
- $(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$;
- $(\rho, i) \models \varphi_1 \vee \varphi_2$ iff $(\rho, i) \models \varphi_1$ or $(\rho, i) \models \varphi_2$;
- $(\rho, i) \models \varphi_1 \rightarrow \varphi_2$ iff $(\rho, i) \models \varphi_1$ then $(\rho, i) \models \varphi_2$;
- $(\rho, i) \models \Box_{[t1,t2]}\varphi$ iff $\forall j \geq i : (\rho, j) \models \varphi \wedge \tau_j - \tau_i \in [t1, t2]$;
- $(\rho, i) \models \Diamond_{[t1,t2]}\varphi$ iff $\exists j \geq i : (\rho, j) \models \varphi \wedge \tau_j - \tau_i \in [t1, t2]$;
- $(\rho, i) \models \varphi_1 \mathcal{U}_{[t1,t2]}\varphi_2$ iff $\forall j, i \leq j < k : (\rho, j) \models \varphi_1$ and $\exists k \geq i : (\rho, j) \models \varphi_2 \wedge \tau_j - \tau_i \in [t1, t2]$.

A dynamic slice of a model reduces the size of the model and focuses on its particular use. However, most slice methods run the model and construct the state space from the initial marking, which involves high costs.

The firing interval in a TPN is relative to the time at which the transition is enabled, and it cannot directly provide information about the firing sequences of any two transitions t_i and t_j where $t_i\bullet \neq \bullet t_j$. Considering the TPN shown in Figure 1, transitions t_4 and t_5 can fire after the transition is enabled at times [3], [5] and [2], [4], respectively, but these values cannot inform the firing sequence. Although the state class can provide any firing sequence, none of the values can provide the exact time at which the TPN is executed from the initial marking up to the transition to fire.

The next section proposes a directed graph that provides information about the firing sequence and the global firing time interval. By "global," we mean the accumulated time interval from the initial marking until the transition fire. This graph is called a *firing dependency graph,* and it can represent the behavior of a TPN and estimate the possible global time of a transition to fire. Our algorithm uses it to consider removing TPN elements that may not affect the given criterion.

III. METHODOLOGY

This section proposes a dynamic slicing algorithm for TPNs based on the firing dependency graph. This algorithm can reduce the size of the TPN model based on the MTL properties by eliminating the place and transition sets that are irrelevant to the MTL properties. The process in Figure 2 shows how to slice a TPN based on the MTL properties. First, an FDG is constructed from the TPN model. Subsection III-A describes the firing dependency graph and how it is created. Second, the slicing criterion is extracted based on the MTL properties. Third, the firing node that cannot be reached from the initial marking within the criterion is sliced from the FDG. Finally, the sliced FDG transforms into a sliced TPN based on the MTL properties. Subsection III-B describes our algorithm for how to slice a time Petri net.

A. FIRING DEPENDENCY GRAPH

A firing dependency graph (FDG) is a dependency graph representing the dependencies of the firing transition within a TPN model. Definition 2 defines a firing dependency graph.

Definition 2 (Firing Dependency Graph): Let $TPN = (P, T, B, F, m_0, ST)$ be a time Petri net. Then, a firing

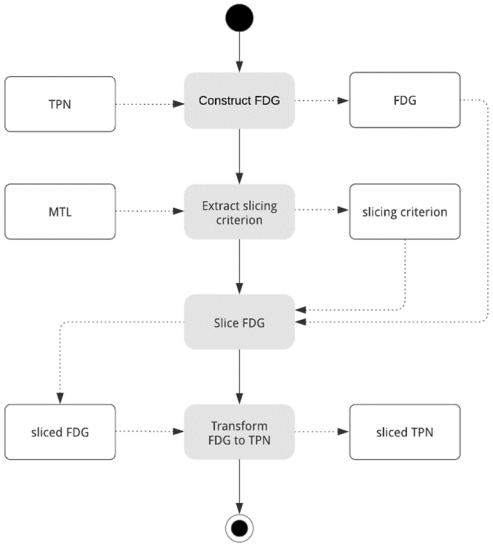

FIGURE 2. Process for slicing the time petri net.

dependency graph is a quadruple $FDG = (FT, E, FT_0, gt)$, where

- FT is a finite set of firing nodes. For a firing node, ft is firing transition t, $t \in T$.
- $E \subseteq FT \times FT$ is a finite set of edges. For an edge, $e = (ft_i, ft_j) \in E, ft_i \neq ft_j$ means that transition ft_j is always enabled after transition ft_i is fired. Therefore, a firing node ft_i is said to be a *predecessor node* of firing node ft_j whereas a firing node ft_i is said to be a *successor node* of firing node ft_j, $\bullet ft$ denotes the set of *predecessor nodes* of firing node ft, and $ft\bullet$ denotes the set of *successor nodes* of firing node ft.
- FT_0 is the set of initial firing nodes. Firing node ft is said to be an *initial firing node* if $t \in En(m_0)$
- $gt : FT \to \mathbb{Q}^* \times (\mathbb{Q}^* \cup \{\infty\})$ is the global firing function that defines the earliest and latest global firing times of each transition, where $\mathbb{Q}^*$ is the set of positive rational numbers. For firing node ft, $gt(ft) = [egt(ft), lgt(ft)]$, where $egt\ (ft)$ and $lgt(ft)$ are the earliest and latest global firing times of transition ft, respectively.

Furthermore, when considering the firing conditions of a transition, we assume that transition ft was enabled at time τ and that transition ft fired at time $\tau + \theta$ if and only if the following conditions hold: (1) According to the enabling rules, ft is enabled at time τ. (2) Relative firing time θ is between $est\ (ft)$ and lst of all enabled transitions at marking m [31]. Then, the global firing time of firing node ft is defined as

$$gt\ (ft) = gt\_en\ (ft) + rt(ft) \tag{1}$$

where $gt(ft)$ is the global firing time interval in which the TPN runs from the initial node to current node ft; $gt\_en(ft)$ is the global enabled time at which the TPN runs from the initial marking up until the time at which transition ft is enabled; and $rt(ft)$ is the relative firing time of transition ft, defined as

$$rt(ft) = \begin{cases} [est\ (ft)\ , min\left\{lst\ \left(t_{cf}\right)\right\}], & case1 \\ st\ (ft)\ , & case2 \end{cases} \tag{2}$$

Cases 1 and 2 are defined as follows:

case 1. If t is the conflict transition and $t_{cf} \in CF\ (t)$, where $CF(t)$ is the set of conflicting transitions with transition t;

case 2. Otherwise.

According to (1), we divide the firing nodes into four groups: (1) The *initial firing node* where the initial marking m_0 enables the transition. (2) The *synchronized firing node* where the transition is synchronized. (3) The *merged firing node* where the transitions of $\bullet ft$ are merged. (4) The *general firing node* with only one predecessor node. The global firing time gt is calculated as follows:

$$gt\ (ft) = \begin{cases} rt\ (ft)\ , \\ \quad case1 \\ \left[\max\{egt(t_k)\}, \max\{lgt(t_k)\}\right] + rt\ (ft)\ , \\ \quad case2 \\ \left[\min\{egt(t_k)\}, \max\{lgt(t_k)\}\right] + rt\ (ft)\ , \\ \quad case3 \\ gt\ (\bullet ft) + rt(ft), \\ \quad case4 \end{cases} \tag{3}$$

Cases 1 to 4 are, respectively:

case 1. If $ft \in FT_0$;

case 2. If t is synchronized transition and $t_k \in \bullet ft$;

case 3. If $t_k \in \bullet ft$ and t_k are the merge transition;

case 4. Otherwise.

Algorithm 1 illustrates how to construct the FDG from $TPN = (P, T, B, F, m_0, st)$. The incident matrix represents the TPN and FDG in this study.

Algorithm 1 : Construct a Firing Dependency Graph

Input: $TPN = (P, T, B, F, m_0, st)$ is a Time Petri net
Output: $FDG = \left(FT, E, ft_0, gt\right)$ is a firing dependency graph

1 Create the set of firing nodes $FT = T$;
2 Create the input flow matrix BM;
3 Create the output flow matrix FM;
4 Create the firing transition relation matrix $E = FM \times BM$;
5 Create the set of initial nodes $ft_0 = En(m_0)$;
6 Initial the set of nodes $queT = ft_0$;
7 **for each** $ft \in queT$ **do**
8 Calculate $gt(ft)$, the vector of the global firing time of transition t.
9 **if** ft *has successornodes* **then**
10 queT $= queT \cup$ *successor* (ft);
11 **end**
12 $queT = queT - \{ft\}$;
13 **end**
14 **return** $FDG = \left(FT, E, ft_0, gt\right)$

According to Algorithm 1, Line 1 creates all the firing nodes of the *FDG*. Then, Lines 2 to 4 construct all the edges of the *FDG* by multiplying an input flow matrix by an output flow matrix. Next, Line 5 creates all initial firing nodes. Finally, Lines 6 to 13 calculate the global firing time of each firing node from the initial firing nodes. In addition, regarding the time complexity, Algorithm 1 is a breadth-first traversal algorithm. Therefore, the boundary of the time complexity is the number of elements in the original time Petri net.

Therefore, the time complexity of Algorithm 1 is $O(|T|^2|P|)$, where $|T|$ is the number of transitions and $|P|$ is the number of places in the unsliced time Petri net.

Example 1: Consider the *TPN* model shown in Figure 1, where transition $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$. The initial marking $m_0 = (1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0)^T$, and $En(m_0) = \{`t1', `t6'\}$.

The input flow matrix *BM* and output flow matrix *FM* are as follows:

$$BM = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

$$FM = \begin{bmatrix} 0&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix}$$

According to Algorithm 1, firing node $FT = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$.

Edge $E = FM \times BM = \{(t_1, t_2), (t_1, t_3), (t_2, t_4), (t_3, t_5), (t_4, t_7), (t_5, t_7), (t_6, t_8), (t_7, t_8)\}$.

Initial firing nodes $FT_0 = t_1, t_6$.

According to (3), we calculate the global firing time gt of each firing node from the initial firing nodes such that:

$$gt(t_1) = rt(t_1) = [3, 6].$$
$$gt(t_6) = rt(t_6) = [1, 8].$$

Next, firing transition $t1$ causes transitions $t2$ and $t3$ such that t_2 and t_3 are the successor nodes of t_1, where

$$gt(t_2) = gt(t_1) + rt(t_2) = [3, 6] + [1, 4] = [4, 10],$$
$$gt(t_3) = gt(t_1) + rt(t_3) = [3, 6] + [2, 5] = [5, 11].$$

Similarly, we derive t_4 and t_5 as

$$gt(t_4) = gt(t_2) + rt(t_4) = [4, 10] + [3, 5] = [7, 15].$$
$$gt(t_5) = gt(t_3) + rt(t_5) = [5, 11] + [2, 4] = [7, 15].$$

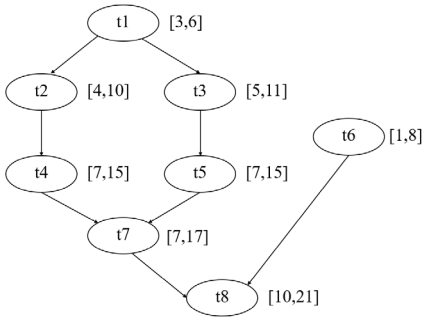

FIGURE 3. The firing dependency graph of the *TPN* in Figure 1.

Firing node t_7 is a synchronized firing node. According to (3), $\bullet t_7 = \{t_4, t_5\}$. Then,

$$\begin{aligned} gt(t_7) &= [\max\{egt(\bullet t_7)\}, \max\{lgt(\bullet t_7)\}] + rt(t_7)\\ &= [\max\{egt(t_4), egt(t_5)\}, \max\{lgt(t_4), lgt(t_5)\}]\\ &\quad + rt(t_7)\\ &= [max\{7, 7\}, max\{15, 15\}] + [0, 2]\\ &= [7, 15] + [0, 2] = [7, 17] \end{aligned}$$

We can generate a firing-dependency graph following this step, as shown in Figure 3.

The FDG can represent the TPN's behavior and the global firing time of each transition, which is the possible accumulated time interval from the initial marking until the transition fired, as will be shown in Theorem 1. To facilitate our description, we illustrate the firing schedule that leads the TPN from the initial ft_0 to ft_n by firing $t_0 t_1 \ldots t_n$ using $ft_0 ft_1 \ldots ft_n$.

Theorem 1: Let ft_i be a firing node from ft_0. Then, gt_i is the global firing time of t_i.

Proof: From the precondition, we know that there must be a firing schedule starting with ft_0 and ending with ft_i, that is, $ft_0 ft_1 \ldots ft_{i-1} ft_i$. We apply mathematical induction to exponent i.

For the basic step ($i = 0$), ft_0 is the initial firing node. We have $\forall t_j \in En(m_0)$, the global enabled time of $t_j = [0, 0]$. Therefore, gt_j is the exact relative firing time and global firing time of t_j. Therefore, this assumption is true for $i = 0$.

For the inductive step, we assume that the assertion holds for all positive integers k. Furthermore, we assume that gt_k is the global firing time of t_k, which is a firing transition at ft_k. Under this assumption, we must show that gt_{k+1} is the global firing time of t_{k+1}, a firing transition at ft_{k+1}. We now apply this to three different cases of ft_{k+1}.

Case 1. ft_{k+1} is a general firing node that has only one predecessor node.

$$\begin{aligned} \text{According to (1), } gt_{k+1} &= gt\_en(ft_{k+1}) + rt(ft_{k+1})\\ &= gt_k + rt(ft_{k+1}) \end{aligned}$$

where gt_k is the global firing time of ft_k, which is also the global enabled time of t_{k+1}. Because firing a transition takes no time, $rt(ft_{k+1})$ is the relative firing time of t_{k+1}. Therefore, $gt_k + rt(ft_{k+1})$ is the global firing time interval of t_{k+1}.

Case 2. ft_{k+1} is the synchronized firing node in which the transition is synchronized. We have $\forall t_j \in \bullet ft$.

$$\begin{aligned} \text{According to (1),}\ gt_{k+1} &= gt\_en\,(ft_{k+1}) + rt(ft_{k+1}) \\ gt_{k+1} &= \left[max\{egt(ft_j)\}, max\{lgt(ft_j)\}\right] \\ &\quad + rt(ft_{k+1}) \\ gt_{k+1} &= [max\left\{egt\left(ft_j\right)\right\} + ert\left(ft_{k+1}\right), \\ &\quad \times max\{lgt(ft_j)\} + lrt\,(ft_{k+1})] \end{aligned}$$

where $\left[max\{egt(ft_j)\}, max\{lgt(ft_j)\}\right]$ is the global enabled time of ft_{k+1}. Because ft_{k+1} is enabled only when any transition ft_j is fired and $ert\,(ft_{k+1})$ is the earliest relative firing time of ft_{k+1}, $max\left\{egt\left(ft_j\right)\right\} + ert\,(ft_{k+1})$ is the earliest global firing time of ft_{k+1}. Additionally, $lrt\,(ft_{k+1})$ is the latest relative firing time of ft_{k+1}; therefore, $max\left\{lgt\left(ft_j\right)\right\} + lrt\left(ft_{k+1}\right)$ is the latest global firing time of ft_{k+1}. Therefore, gt_{k+1} is the global firing time interval of t_{k+1}.

Case 3. ft_{k+1} is the merged firing node, where the transitions of $\bullet ft$ are merged transitions. The transition ft_{k+1} is enabled when a transition in $\bullet ft$ is fired. We have $\forall t_j \in$ the domain *of* $\bullet ft$.

$$\begin{aligned} \text{According to (1),}\ gt_{k+1} &= gt\_en\,(ft_{k+1}) + rt(ft_{k+1}) \\ gt_{k+1} &= \left[min\{egt(ft_j)\}, max\{lgt(ft_j)\}\right] \\ &\quad + rt(ft_{k+1}) \\ gt_{k+1} &= [min\left\{egt\left(ft_j\right)\right\} + ert\,(t_{k+1}), \\ &\quad \times max\{lgt(ft_j)\} + lrt\left(ft_{k+1}\right)] \end{aligned}$$

where $\left[min\{egt(ft_j)\}, max\{lgt(ft_j)\}\right]$ is the earliest global enabled time of ft_{k+1}. As ft_{k+1} is enabled when a transition ft_j is fired and $ert\,(ft_{k+1})$ is the earliest relative firing time of ft_{k+1}, $min\left\{egt\left(ft_j\right)\right\} + ert\left(ft_{k+1}\right)$ is the earliest global firing time of ft_{k+1}. Additionally, $lrt\,(ft_{k+1})$ is the latest relative firing time of ft_{k+1}; therefore, $max\left\{lgt\left(ft_j\right)\right\} + lrt\left(ft_{k+1}\right)$ is the latest global firing time of ft_{k+1}. Therefore, the global firing time interval of ft_{k+1} is gt_{k+1}.

Hence, the theorem holds.

The proof shows that if the sequence $t_0 t_1 \ldots t_{i-1}$ is fired, then the current time must be in gt_i, and any time τ in gt_i must be able to fit into the firing sequence $t_0 t_1 \ldots t_{i-1}$ and end at time τ and t_i. Therefore, gt_i gives the exact global time at which transition t_i fires.

Corollary 1: Let ft_i and ft_j be the two firing nodes of an *FDG*. If gt_i occurs before gt_j, then ft_i always fires before ft_j fires.

Proof: In Allens' Interval Algebra [32], if gt_i is before gt_j, then $egt\,(ft_i) < lgt\left(ft_i\right) < egt\left(ft_j\right) < lgt(ft_j)$. $lgt\left(ft_i\right)$ is the latest global firing time of t_i, and $egt\left(ft_j\right)$ is the earliest global firing time of ft_j; therefore, ft_j must fire before or at the same time as $lgt\,(ft_i)$, and ft_j cannot fire before time $egt\left(ft_j\right)$. Therefore, ft_i always fires before ft_j fires.

Corollary 2: Let ft_i and ft_j be the two firing nodes of an *FDG*. If ft_j is reachable from ft_i, then the time interval in which the TPN runs (from ft_i to ft_j) is $gt_j - gt_i$.

Proof: With a TPN starting from $\bullet t_i$ at time 0, ft_i will have fired at the global time [a, b], and then ft_j will fire globally [a + c, b + d]. The time interval from which the TPN runs from ft_i to ft_j is [a + c, b + d] – [a, b] = [c, d]. Furthermore, if the TPN starts from $\bullet t_i$ at time [x, y], ft_i fires globally [x + a, y + b], and ft_j fires globally [x + a + c, y + b + d]. The time interval from which the TPN runs from ft_i to ft_j is [x + a + c, y + b + d] − [x + a, y + b] = [c, d].

The time interval from which the TPN runs from ft_i to ft_j is independent of the starting time. According to Theorem 1, the time interval is $gt_j - gt_i$.

Example 2: Consider the *FDG* shown in Figure 3. Suppose that we have a time constraint that determines the transitions in which ft_1, ft_2, ft_3 and ft_6 must fire within five time units from the initial marking. We then know that the constraint is satisfied.

Example 3: Consider the *FDG* shown in Figure 3. We found that ft_8 can be reached from ft_3 via many firing sequences. The time interval proceeds via each firing sequence within [10, 21] – [5, 11] = [5, 10].

Corollaries 1 and 2 can be used for the timeless analysis of whether a transition can fire with time constraints. Corollary 1 is used to find the reachable firing node within the timing constraint. Corollary 2 is used to establish a quantitative timing relationship between any two reachable transitions.

The following section proposes a way to reduce the size of an FDG based on the time criterion to reduce the size of a TPN model.

B. SLICING OF THE TIME PETRI NET MODEL

Based on the FDG, we can derive a sliced net under any firing node criterion. The algorithm starts with the transition criterion, according to which the successor node of the criterion of the place travels the FDG. Then, we derive a reachable firing node within the global firing time of the transition criterion. Thus, a subgraph of the FDG represents the behavior of a sliced TPN under the initial marking, and its firing nodes represent the transitions of a sliced TPN. Algorithm 2 describes our slicing TPN as slicing a TPN based on an FDG.

According to Algorithm 2, Lines 1 to 2 construct the firing dependency graph (FDG) from the TPN model computed according to Algorithm 1. Then, Lines 3 to 4 extract the slicing firing node criterion (*tCrit*) from the MTL formula by deriving the criterion from all propositions of the MTL. Next, Lines 5 to 27 slice the *FDG* based on *tCrit*. Lines 5 to 10 create the initial firing nodes that run into *tCrit*, Lines 11 to 19 remove the firing nodes that cannot run into *tCrit*, Lines 20 to 23 remove the firing nodes that always fire after *tCrit* fires, and Lines 24 to 27 add the firing nodes that are elements of the conflict transition set in FT'. The final graph contains all the firing nodes representing the transition set of the sliced TPN. Finally, Lines 28 to 33 construct the sliced TPN from the sliced FDG as follows: Line 28 initializes the transitions (T') in the firing node of the sliced FDG; Line 29 initializes the input places (P') of the transitions in both TPN' and TPN; and Line 30 initializes the backward

Algorithm 2 : Slicing Time Petri Net

Input: $TPN = (P, T, B, F, m_0, st)$ is a time Petri net
MTL is a matrix temporal logic formula
Output: $TPN' = (P', T', B', F', m_0', st')$ is a sliced TPN

1 Construct the firing dependency graph $FDG = (FT, E, gt, ft_0)$
2 Initialize the set of firing nodes $FT' = FT$;
3 Initialize the place criterion $pCrit =$ set of propositions of MTL
4 Initialize the firing node criterion set $tCrit = \bullet pCrit$;
5 Initialize the set of the initial firing nodes $ft_0' = \{\}$;
6 **for each** $t \in ft_0$ **do**
7 **if** $tCrit$ is *reachable from* t **then**
8 $ft_0' = ft_0' \cup \{t\}$;
9 **end**
10 **end**
11 Initialize the set of nodes $queT = (ft_0 - ft_0') \cup tCrit$;
12 $FT' = FT' - (ft_0 - ft_0')$;
13 **for each** $ft \in queT$ **do**
14 **while** ft *has successornodes* **then**
15 $FT' = FT' -$ *successor* (ft);
16 queT $= queT \cup$ *successor* (ft);
17 **end**
18 $queT = queT - \{ft\}$;
19 **end**
20 Calculate $max\_gt(tCrit)$, the maximum global firing time of $tCrit$;
21 **for each** $ft' \in FT'$ **do**
22 **if** $tCrit$ *is unreachable from* ft' **and** $gt(ft') > max\_gt\_tCrit$ **then**
23 $FT' = FT' - ft'$
24 **else if** ft'*is conflict transition* **then**
25 $FT' = FT' \cup CF(ft')$
26 **end**
27 **end**
28 $T' = FT'$;
29 $P' = \{t \in T \cap T' | p = \bullet t\}$
30 $B' = \{p \in P \cap P', t \in T \cap T' | b(p, t)\}$
31 $F' = \{p \in P \cap P', t \in T \cap T' | f(p, t)\}$
32 $st' = \{t \in T \cap T' | st(t)\}$
33 $m_0' = \{p \in P \cap P' | m(p)\}$
34 **return** $TPN' = (P', T', B', F', m_0', ST')$

incidence function (B') of TPN, the places and transitions of which are in both TPN' and TPN.

Similarly, Line 31 initializes the forward incidence function (F') of TPN, the places and transitions of which are in both TPN' and TPN. Line 32 initializes the static firing interval (st'). In TPN, st' is the static firing interval of the transitions in both TPN' and TPN. Finally, Line 33 initializes the initial marking (m_0') of TPN, the places of which are in both TPN' and TPN.

As for the time complexity of the algorithm, Algorithm 2 constructs a firing dependency graph and slicing FDG. In addition, when considering the slicing FDG, this algorithm is a breadth-first traversal algorithm. Hence, the boundary of the time complexity is the number of elements in the unsliced FDG. The number of firing nodes and the number of edges of FDG equal the number of transitions and the number of forward incidence functions of TPN, respectively. Therefore, the time complexity related to slicing the FDG algorithm is

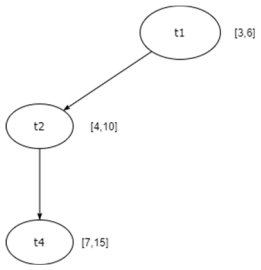

FIGURE 4. The SDG of the TPN in Figure 1, where the place criterion is p7.

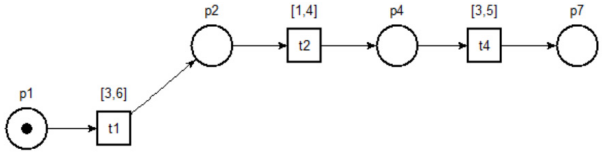

FIGURE 5. *TPN'*, where the place criterion is *p7*.

$O(|T| + |F|)$, where $|T|$ is the number of transitions and $|F|$ is the number of forward incidence functions in the unsliced TPN. Furthermore, the time complexity of constructing a firing dependency graph is $O(|T|^2\,|P|)$.

Consequently, the time complexity of Algorithm 2 is $O(|T|^2\,|P|+|T|+|F|)$, where $|T|$ is the number of transitions, $|P|$ is the number of places, and $|F|$ is the number of forward incidence functions in the unsliced time Petri net.

Example 4: Consider the TPN model, as shown in Figure 1, where the place criterion is $p7$. We can construct SDG, as shown in Figure 4; and TPN', as shown in Figure 5.

IV. CASE STUDY

This section presents an experiment to verify the timing properties of a tactic anti-air command and control (C2) system derived from [33]. The C2 system consists of one C2 center, two subcenters, two air radar groups, and two firing units.

As for the operations of the system, first, each air radar group senses air targets to fuse the data at the data processor and then sends messages to the corresponding subcenter. Second, after each subcenter receives the message from its radar group, it conducts its commands and sends them to the C2 center. Third, when the messages arrive at the C2 center from two subcenters, situational assessment is performed at the two intelligence seats, and a scheme is worked on at the decision-making seat before sending the results to the two subcenters. Fourth, the subcenter fuses the message with the related data and then sends the results to the firing units upon receiving the command from the C2 center. Finally, when the firing unit receives an assignment from its subcenter, it conducts the assessment and feeds the results back to its corresponding subcenter. Table 1 shows the duration of each operation. The TPN model represents the structure and behavior of this system, as shown in Figure 6; and Tables 1 and 2 provide transition and place descriptions, respectively.

Furthermore, the two systems considered for the slicing criterion in this example are as follows:

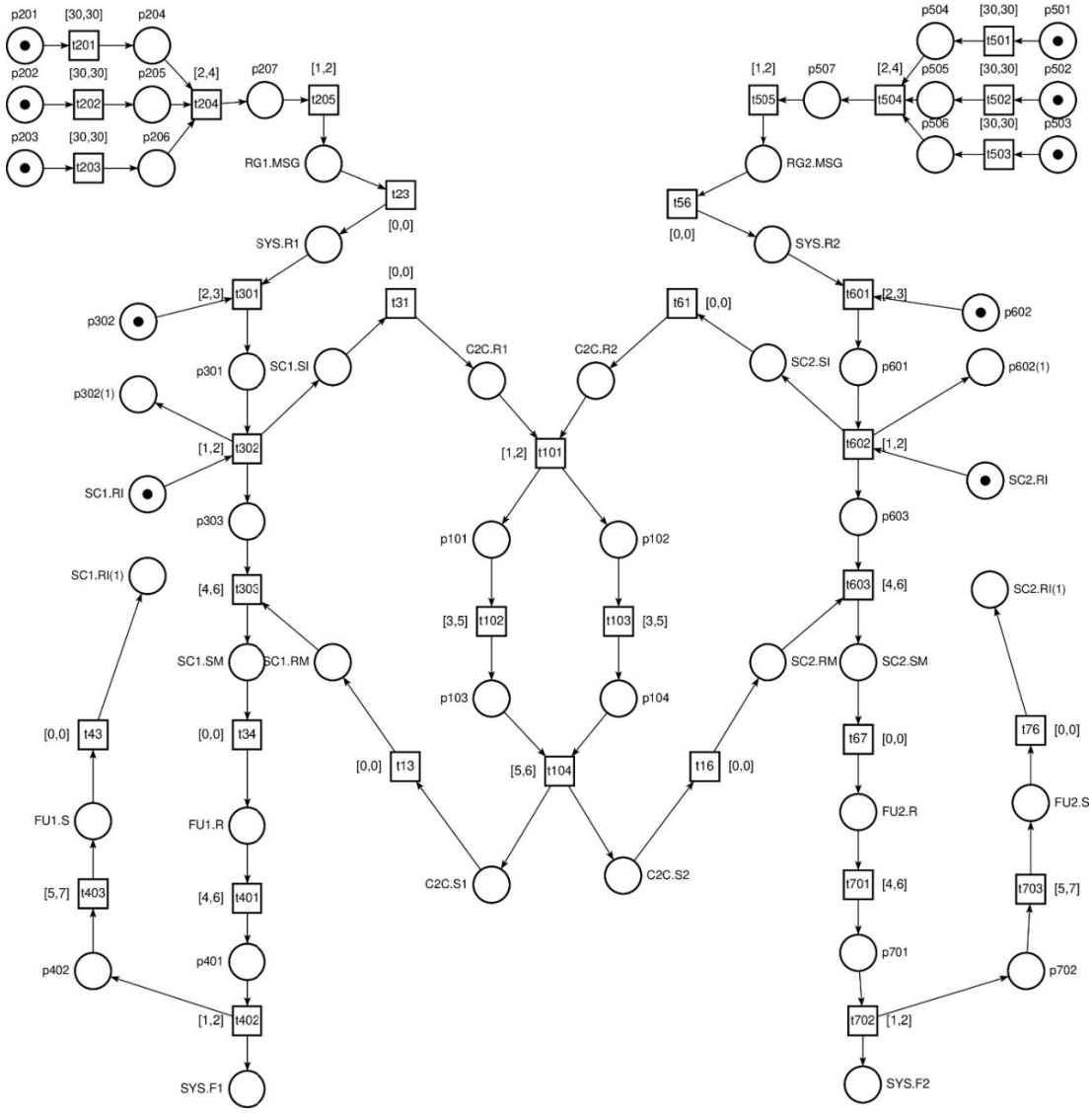

FIGURE 6. The TPN of the tactical anti-air C2 system.

TABLE 1. The transition description.

Transition	Description	Timing Interval
t101	Dispatch intelligence messages to two staff seats	[1,2]
t102, t103	Two staff seats conduct a situational assessment	[3,5]
t104	The top commander seat conducts information fusion and combat planning	[5,6]
t13, t16, t31, t61	The connectors among the C2 centers and subcenter 1 (subcenter 2)	[0,0]
t201, t202, t203, t501, t502, t503	The radar senses	[30,30]
t204, t504	The processor fuses data	[2,4]
t205, t505	The processor codes the fused data	[1,2]
t23, t56	The connectors among the air radar group 1 and subcenter 1	[0,0]
t301, t601	Subcenter 1 (subcenter 2) conducts target discrimination, identification, and tracking	[2,3]

Property 1: Each air radar group sends the target information to its subcenter, which takes 40 time units. The MTL representation of this can be expressed as follows:

$$\Box(((p201 \wedge p202 \wedge p203) \rightarrow (\Diamond_{\leq 40} RG1.MSG)) \wedge ((p501 \wedge 502 \wedge p503) \rightarrow (\Diamond_{\leq 40} RG2.MSG)))$$

Property 2: The C2 center groups the messages from all subcenters and sends back results. The entire processing time must be less than or equal to 22 time units. The MTL is expressed as follows:

$$\Box((C2C.R1 \wedge C2C.R2) \rightarrow (\Diamond_{\leq 22}(C2C.S1 \wedge C2C.S2))$$

TABLE 2. The place description.

Place	Description
p101, p102	Ready for situation assessment
p103, p104	Ready for fusion and combat planning
p201, p202, p203, p501, p502, p503	Radar ready to sense air targets
p204, p205, p206, p504, p505, p506	Radar data for fusion
p207, p507	Fused radar data
p301, p601	Waiting for the evaluation of a threat
p302, p302(1), p602, p602(1)	Intelligence seat available
p303, p603	Waiting for a fire assignment
p401, p701	Ready to send a fire command
p402, p702	Waiting for damage assessment
C2C.R1, C2C.R2	C2 center receives a message from subcenter 1 (subcenter 2)
C2C.S1, C2C.S2	C2 center ready to send the command to subcenter 1 (subcenter 2)
FU1.R, FU2.R	Firing unit 1 (firing unit 2) receives the command from subcenter 1 (subcenter 2)
FU1.S, FU2.S	Firing unit 1 (firing unit 2) is ready to send the result of the damage assessment to subcenter 1 (subcenter 2)
RG1.MSG, RG2.MSG	Radar group 1 (radar group 2) is ready to send the target message to subcenter 1 (subcenter 2)
SC1.RI, SC2.RI, SC1.RI (1), SC2.RI (1)	Subcenter 1 (subcenter 2) receives the damage assessment result from firing unit 1 (firing unit 2)
SC1.RM, SC2.RM	Subcenter 1 (subcenter 2) receives the command from the C2 center
SC1.SM, SC2.SM	Subcenter 1 (subcenter 2) is ready to send the command to firing unit 1 (firing unit 2)
SYS.F1, SYS.F2	A combat command is sent to firing unit 1 (firing unit 2)
SYS.R1, SYS.R2	A message arrives from air radar group 1 (air radar group 2)

In the experiment, we slice the time Petri net based on two properties according to Algorithm 2. Additionally, we compared the structural model, state space class, and the results of the MTL satisfaction check between the sliced and unsliced TPNs. The comparison of the structural model focuses on the total number of places, the number of transitions, and the number of arcs that link these places and transitions. Alternatively, comparing the state–space class focuses on the number of states and the number of edges that connect those states. In this study, the model checker used to construct the structure and state space models is the time Petri net analyzer (TINA) toolbox, dependent on the LTL properties. As for the MTL satisfaction check results, we split the MTL properties into two categories: the LTL and timing properties of the C2 system. Then, we verify the LTL properties using the TINA model checker and analyze the timing properties according to corollary two.

As for the experimentation, applying Algorithm 2 to slice the TPN based on two properties gives the following FDG:

FT = {t101, t102, t103, t104, t13, t16, t201, t202, t203, t204, t205, t23, t301, t302, t303, t31, t34, t401, t402, t403, t43, t501, t502, t503, t504, t505, t56, t601, t602, t603, t61, t67, t701, t702, t703, t76}.

gt = {gt(t101) = [37, 43], gt(t102) = [40, 48], gt(t103) = [40, 48], gt(t104) = [45, 54], gt(t13) = [45, 54], gt(t16) = [45, 54], gt(t201) = [30, 30], gt(t202) = [30, 30], gt(t203) = [30, 30], gt(t204) = [32, 34], gt(t205) = [33, 36], gt(t23) = [33, 36], gt(t301) = [35, 39], gt(t302) = [36, 41], gt(t303) = [49, 60], gt(t31) = [36, 41], gt(t34) = [49, 60], gt(t401) = [53, 66], gt(t402) = [54, 68], gt(t403) = [59, 75], gt(t43) = [59, 75], gt(t501) = [30, 30], gt(t502) = [30, 30], gt(t503) = [30, 30], gt(t504) = [32, 34], gt(t505) = [33, 36], gt(t56) = [33, 36], gt(t601) = [35, 39], gt(t602) = [36, 41], gt(t603) = [49, 60], gt(t61) = [36, 41], gt(t67) = [49, 60], gt(t701) = [53, 66], gt(t702) = [54, 68], gt(t703) = [59, 75], gt(t76) = [59, 75]}.

Furthermore, when considering the sliced TPN based on property 1 and its capacity for satisfaction, the slicing criterion is {RG1.MSG, RG2.MSG}, and the time delay from the initial marking to RG1.MSG and RG2.MSG is 40 time units. Thus, the transition criterion is {t205, t505}. Our slicing algorithm can remove all firing nodes that always fire after the transition criterion. Figure 7 shows the sliced TPN based on property 1, and Table 3 presents the number of places and transitions of unsliced and sliced TPNs. Finally, considering MTL satisfaction, the result for the LTL property is true, as shown in Table 4. Furthermore, when considering the timing property, the time interval in which the transition criterion can be reached from the initial marking is [33], [36], so the timing property is satisfied. Therefore, property 1 of the system was satisfied.

Similarly, slicing the TPN based on property 2 yields the following results: the slicing criterion is {C2C.R1, C2C.R2, C2C.S1, C2C.S2}; therefore, the transition criterion is {t31, t61, t104}. The time delay from {C2C.R1, C2C.R2} to {C2C.S1, C2C.S2} was 22 time units. All firing nodes always fire after the transition criterion is removed. Figure 8 shows the slicing of the TPN based on property 2. As a result, the TPN model size is summarized in Table 3. As for the satisfaction of the MTL, the LTL property is satisfied, as shown in Table 4. Additionally, according to Corollary 2, the time interval from {C2C.S1, C2C.S2} to {C2C.R1, C2C.R2} is the time interval from which the TPN runs from {t31, t61} to t104 and is [45, 54] - [36, 41] = [9, 13], so the timing property is satisfied. Therefore, the system property was satisfied.

In addition, we combine two properties to obtain the overall system property. The slicing criterion is {RG1.MSG, RG2.MSG, C2C.R1, C2C.R2, C2C.S1, C2C.S2}, so the

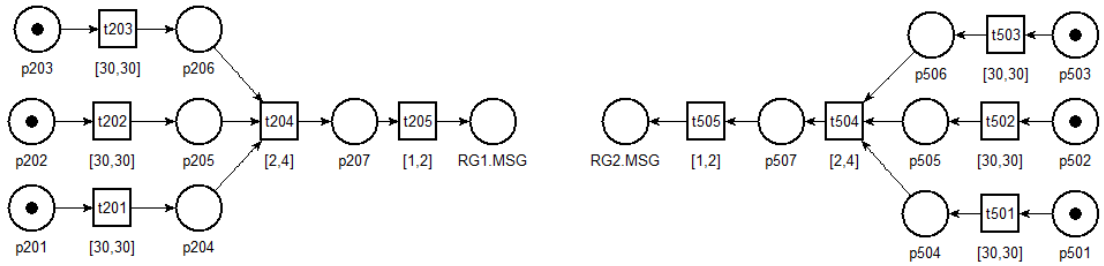

FIGURE 7. The *TPN'* where the criterion is property 1.

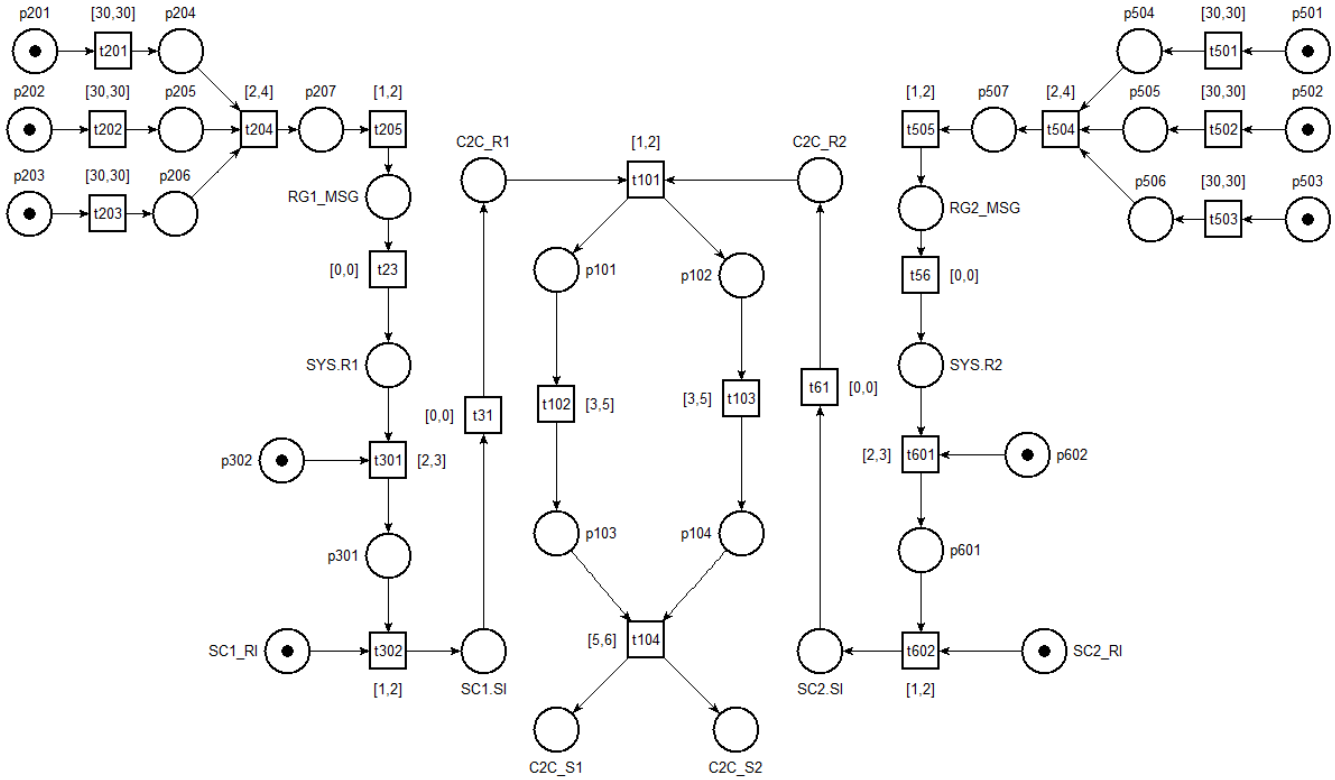

FIGURE 8. The *TPN'* where the criterion is property 2.

transition criterion is {t205, t505, t31, t61, t104}. The time delay from the initial marking to RG1.MSG and RG2.MSG is 40 time units, and from {C2C.R1, C2C.R2} to {C2C.S1, C2C.S2} was 22 time units. Our slicing algorithm can remove all firing nodes that always fire after the transition criterion, which is the same as the sliced of the TPN based on property 2, as shown in Figure 8. According to the satisfaction of both LTL and timing property, the system property was satisfied.

Table 3 shows the number of places and transitions of unsliced and sliced TPNs. When comparing the sizes of the sliced and unsliced TPN models, the sliced TPN is smaller than the unsliced TPN.

Table 4 shows the results of state space analysis by the TINA model checker. In terms of the number of states, the state space size of the sliced TPN is smaller than that of the unsliced TPN; furthermore, the sliced TPN still preserved all properties of the LTL satisfaction results of the unsliced TPN, and the sliced TPN consumed less CPU time.

Suppose that the slicing criterion is the output place set of all transitions that fired in the same order. We then slice the

TABLE 3. The structural models of unsliced and sliced TPNs.

Time Petri net	No. of Places	No. of Transitions
Unsliced time Petri net	54	36
Sliced based on property 1	16	10
Sliced based on property 2	34	22
Sliced based on system property	34	22

TABLE 4. The state space analysis of unsliced and sliced TPNs.

Time Petri net	Unsliced	Sliced based on property 1	Sliced based on property 2	Sliced based on system property
the number of states	253	77	176	176
boundedness	yes	yes	yes	yes
liveness	no	no	no	no
the number of live states	1	1	1	1
the numbers of deadlock states	1	1	1	1
LTL satisfaction (property 1)	true,	true,	true,	true,
LTL satisfaction (property 2)	true,	-	true,	true,
LTL satisfaction (property 1 and 2)	true,	-	true,	true,

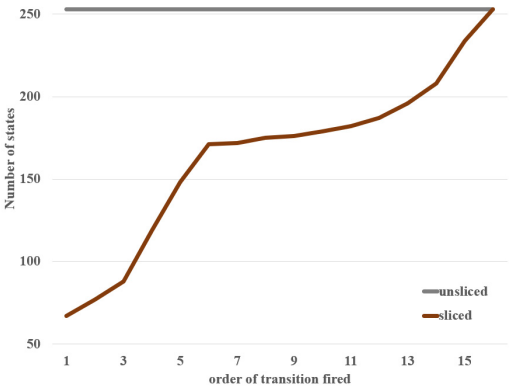

FIGURE 9. Comparison of the number of states of the sliced TPN with that of the unsliced TPN. On the X-axis is the order of the transitions fired in ascending order.

TPN based on each slicing criterion and obtain a plot for the number of states of the sliced TPN, as shown in Figure 9.

Figure 9 compares the number of states sliced from the first transition fired to the last transition fired with the number of states of the unsliced TPN. According to the graph, the state space size of the sliced TPN increases considerably in proportion to the order of the transitions fired and stays the same at the size of the unsliced TPN at the last transition fired.

According to the study findings, our slicing algorithm may reduce the size of the TPN for a software system with target MTL properties preserved as the unsliced TPN decreases the state space size. However, the sliced TPN that concerns the target properties can vary with the MTL properties. Moreover, the order of the transitions fired can considerably influence any reductions. The early order of the transitions fired is attached to the criterion and allows for greater reductions while the last order of the transitions fired may not reduce the size of the sliced TPN.

Additionally, our slicing algorithm is based on a firing dependency graph that represents the overall behavior of a TPN, and it can be possible to compute the global firing time of the criteria. Therefore, our slicing algorithm may slice a TPN where several MTL formulas do not recompute the FDG.

V. CONCLUSION

This paper proposes an alternative dynamic TPN slicing algorithm based on a firing dependency graph, representing the global firing time interval of transitions, to reduce the TPN elements irrelevant to the MTL properties' criterion. Furthermore, the sliced TPN that was obtained only ensured that it had fewer enabled transitions than the unsliced TPN and preserved all the properties of the unsliced TPN.

Our slicing algorithm does not replace traditional model checking. However, we propose an alternative to verify the TPN model to possibly reduce the size of a TPN; and then the sliced TPN, which performs equivalently to the unsliced TPN, is the input of the traditional model checking. Although our algorithm had a cost to reduce the model, it might be helpful for model checking to generate state space. For example, suppose that massive real-time systems with the target MTL formula might cause state space explosion resulting in the inability to conduct model checking. Perhaps we can remove a sufficient number of the transitions that do not affect the state space to prevent state space explosion, and then the model check can be verified.

As the slicing algorithm considers only the propositions of MTL properties, some state space classes may not be reachable from their initial states. Therefore, it is recommended that further research with time constraints of the MTL properties to account for the sliced TPN be conducted. Furthermore, the approach may be applicable to a high-level Petri net, and thus more complex systems with data and variables can be verified.

REFERENCES

[1] B. Berthomieu and M. Menasche, "An enumerative approach for analyzing time Petri nets," in *Proc. IFIP 9th World Comput. Congr.*, Paris, France, 1983, pp. 832–843.

[2] R. Hadjidj and H. Boucheneb, "Efficient reachability analysis for time Petri nets," *IEEE Trans. Comput.*, vol. 60, no. 8, pp. 1085–1099, Aug. 2011.

[3] K. Klai, N. Aber, and L. Petrucci, "Verification of reachability properties for time Petri nets," in *Proc. 7th Int. Workshop Reachability Problems*, Uppsala, Sweden, 2013, pp. 159–170.

[4] A. Bouajjani, S. Tripakis, and S. Yovine, "On-the-fly symbolic model checking for real-time systems," in *Proc. Real-Time Syst. Symp.*, San Francisco, CA, USA, Dec. 1997, pp. 25–34.

[5] N. Jbeli, Z. Sbai, and R. B. Ayed, "On the fly model-checking of TPN: $TPN - TCTL^{\Delta}{}_{h}$," in *Trends and Advances in Information Systems and Technologies*, vol. 2, 1st ed. Cham, Switzerland: Springer, 2018, pp. 441–451.

[6] R. Hadjidj and H. Boucheneb, "On-the-fly TCTL model checking for time Petri nets using state class graphs," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4241–4426, Sep. 2009.

[7] E. Lubat, S. D. Zilio, D. L. Botlan, Y. Pencole, and A. Subias, "A new product construction for the diagnosability of patterns in time Petri net," in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Jeju, South Korea, Dec. 2020, pp. 104–109.

[8] K. Wang, H. Boucheneb, K. Barkaoui, and Z. Li, "Towards efficient partial order techniques for time Petri nets," in *Verification and Evaluation of Computer and Communication Systems*. Cham, Switzerland: Springer, 2020, pp. 100–115.

[9] H. Boucheneb and K. Barkaoui, "Delay-dependent partial order reduction technique for real time systems," *Real-Time Syst.*, vol. 54, no. 2, pp. 278–306, Dec. 2017.
[10] F. M. Bønneland, P. G. Jensen, K. G. Larsen, M. Muñiz, and J. Srba, "Start pruning when time gets urgent: Partial order reduction for timed systems," in *Computer Aided Verification*. Cham, Switzerland: Springer, 2018, pp. 527–546.
[11] K. Barkaoui and H. Boucheneb, "On persistency in time Petri nets," in *Formal Modeling and Analysis of Timed Systems*. Cham, Switzerland: Springer, 2018, pp. 108–124.
[12] H. Boucheneb and K. Barkaoui, "Stubborn sets for time Petri nets," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 1, pp. 1–25, Jan. 2015.
[13] P.-A. Bourdil, B. Berthomieu, S. D. Zilio, and F. Vernadat, "Symmetry reduction for time Petri net state classes," *Sci. Comput. Program.*, vol. 132, pp. 209–225, Dec. 2016.
[14] R. H. Sloan and U. Buy, "Reduction rules for time Petri nets," *Acta Inform.*, vol. 33, no. 7, pp. 687–706, 1996.
[15] C. Xia and Z. Liu, "Property preservation of time Petri net reduction," in *Proc. 3rd Int. Symp. Intell. Inf. Technol. Secur. Informat.*, Jian, China, Apr. 2010, pp. 154–159.
[16] M. Sogbohossou and D. Delfieu, "Temporal reduction in time Petri net," in *Proc. 3rd Int. Design Test Workshop*, Monastir, Tunisia, Dec. 2008, pp. 260–265.
[17] J. Wang, Y. Deng, and M. Zhou, "Compositional time Petri nets and reduction rules," *IEEE Trans. Syst. Man, Cybern., B, Cybern.*, vol. 30, no. 4, pp. 562–572, Aug. 2000.
[18] M. Chalupa and J. Strejček, "Evaluation of program slicing in software verification," in *Integrated Formal Methods*. Cham, Switzerland: Springer, 2019, pp. 101–119.
[19] H. Sabouri and M. Sirjani, "Slicing-based reductions for Rebeca," *Electron. Notes Theor. Comput. Sci.*, vol. 260, pp. 209–224, Jan. 2010.
[20] N. Yatapanage, K. Winter, and S. Zafar, "Slicing behavior tree models for verification," in *Theoretical Computer Science*. Berlin, Germany: Springer, 2010, pp. 125–139.
[21] B. Srongsil and W. Vatanawood, "Compositional verification of data invariants in Promela using slicing technique," in *Proc. Int. MultiConf. Eng. Comput. Scientists*, Hong Kong, 2018, pp. 484–488.
[22] L. I. Millett and T. Teitelbaum, "Issues in slicing PROMELA and its applications to model checking, protocol understanding, and simulation," *Int. J. Softw. Tools Technol. Transf.*, vol. 2, no. 4, pp. 343–349, 2000.
[23] R. Hajisheykhi, M. Roohitavaf, A. Ebnenasir, and S. Kulkarni, "A framework for verification of SystemC TLM programs with model slicing: A case study," in *Proc. 53rd Annu. Design Autom. Conf.*, Austin, TX, USA, Jun. 2016, pp. 1–6.
[24] A. Janowska and P. Janowski, "Slicing of timed automata with discrete data," *Fundamenta Informaticae*, vol. 72, pp. 181–195, Jan. 2006.
[25] C. Thrane and U. Sørensen, "Slicing for UPPAAL," in *Proc. Annu. IEEE Student Paper Conf.*, Aalborg, Denmark, Feb. 2008, pp. 1–5.
[26] A. Rakow, "Safety slicing Petri nets," in *Application and Theory of Petri Nets*. Berlin, Germany: Springer, 2012, pp. 268–287.
[27] A. Rakow, "Slicing Petri nets with an application to workflow verification," in *SOFSEM: Theory and Practice of Computer Science*. Berlin, Germany: Springer, 2008, pp. 436–447.
[28] R. Davidrajuh and A. Roci, "Performance of static slicing algorithms for Petri nets," *Int. J. Simul.-Syst., Sci. Technol.*, vol. 20, no. 1, pp. 15.1–15.7, Mar. 2019.
[29] M. Llorens, J. Oliver, J. Silva, S. Tamarit, and G. Vidal, "Dynamic slicing techniques for Petri nets," *Electron. Notes Theor. Comput. Sci.*, vol. 223, pp. 153–165, Dec. 2008.
[30] W. Yu, Z. Ding, and X. Fang, "Dynamic slicing of Petri nets based on structural dependency graph and its application in system analysis," *Asian J. Control*, vol. 17, no. 4, pp. 1403–1414, Jul. 2015.
[31] L. Opova-Zeugmann, *Time and Petri Nets*, 1st ed. Berlin, Germany: Springer, 2013.
[32] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983.
[33] J. Wang, Y. Deng, and G. Xu, "Reachability analysis of real-time systems using time Petri nets," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 30, no. 5, pp. 725–736, Oct. 2000.

P. CHARIYATHITIPONG received the B.Eng. degree in computer engineering from Khon Kaen University, Thailand, and the M.S. degree in software engineering from Chulalongkorn University, Thailand, where she is currently pursuing the Ph.D. degree in computer engineering.

She was a Lecturer in software engineering at Sisaket Rajabhat University, Thailand. Her research interests include formal specifications and software engineering.

W. VATANAWOOD received the B.Eng. degree in computer engineering from Chulalongkorn University, Thailand, in 1985, the M.Sci. degree in computer science from the California State University, in 1989, and the Ph.D. degree in computer engineering from Chulalongkorn University, in 2002.

He is an Associate Professor with the Computer Engineering Department, Chulalongkorn University. His research interests include enterprise software architectural design, application framework, formal specifications, and software engineering.