Performance Analysis Using Timed Petri Nets

by

Rami R. Razouk
Charles V. Phelps

ABSTRACT

Petri Nets have been successfully used to model and evaluate the performance of distributed systems. Several researchers have extended the basic Petri Net model to include time, and have demonstrated that restricted classes of Petri Nets can be analyzed efficiently. Unfortunately, the restrictions prohibit the techniques from being applied to many interesting systems, e.g. communication protocols. This paper proposes a version of timed Petri Nets which accurately models communication protocols, and which can be analyzed using Timed Reachability Graphs. Procedures for constructing and analyzing these graphs are presented. The analysis is shown to be applicable to a larger class of Timed Petri Nets than previously thought. The model and the analysis technique are demonstrated using a simple communication protocol.

Technical Report #206

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

August 24, 1983

Performance Analysis Using Timed Petri Nets

by

Rami R. Razouk
Charles V. Phelps

ABSTRACT

Petri Nets have been successfully used to model and evaluate the performance of distributed systems. Several researchers have extended the basic Petri Net model to include time, and have demonstrated that restricted classes of Petri Nets can be analyzed efficiently. Unfortunately, the restrictions prohibit the techniques from being applied to many interesting systems, e.g. communication protocols. This paper proposes a version of timed Petri Nets which accurately models communication protocols, and which can be analyzed using Timed Reachability Graphs. Procedures for constructing and analyzing these graphs are presented. The analysis is shown to be applicable to a larger class of Timed Petri Nets than previously thought. The model and the analysis technique are demonstrated using a simple communication protocol.

Introduction

As distributed computer systems have become more widely used, interest has increased in techniques and tools which can be used to evaluate their correctness and estimate their performance. Petri Net models have been recommended by many researchers as useful tools for modeling and evaluating real-time systems, distributed systems and communication protocols. The basic Petri Net model, first discussed in [Petri C. 65], is a general model of computation which can be used to model flow of control, concurrency, and synchronization. This basic model can be analyzed to determine if the system being modeled contains deadlocks or undesirable states. Analysis of Petri Nets using reachability graphs, has been used to verify "safeness" issues in communication protocols, e.g. deadlock freeness. The weakness of the basic model is its inability to model data transformations and timing relationships between events. Several extensions have been proposed which model data transformations [Razouk R. 80, Symons F. 80, Berthelot G. 82]. These extensions have been primarily used to ease the task of verifying partial correctness. Other extensions have been proposed which introduce the notion of *time*. Much of this later work [Merlin P. 76, Ramchandani C. 74, Ramamoorthy C. 80, Sifakis J. 77, Zuberek W. 80] has focussed on analyzing system performance. The success of performance analysis work has been limited to a small class of systems which can be modeled using restricted classes of Petri Nets. Timed Petri Nets (and related models) have also been used as the basis for simulation environments [Razouk R. 79, Vernon M 83].

This paper introduces yet another form of Timed Petri Nets which draws from work by other researchers. Our Timed Petri Nets can be used to model communication protocols elegantly, and can yield interesting performance estimates. In section 1 we remind the reader of the basic Petri Net model and we review past work on Timed Petri Nets. Section 2 introduces the Timed Petri Net model used in this work, and demonstrates the usefulness of the extensions in modeling communication protocols. Section 3 presents procedures for constructing and analyzing Timed Reachability Graphs. Finally, section 4 demonstrates the model and the analysis technique by applying them to a simple communication protocol.

1. Petri Nets and Timed Petri Nets

Since many authors have presented formal definitions of Petri Nets, we dispense with much of the formalism and concentrate on giving the reader an intuitive understanding of the model. For a more formal description of Petri Nets, the reader is referred to [Peterson J. 81].

Petri Nets consist of *transitions* (bars) which model events, and *places* (circles) which model conditions. *Edges* which connect places to transitions describe the conditions under which an event can occur (a transition may *fire*). Edges which connect transitions to places describe the conditions which result from the firing of a transition. The instantaneous state of a net is called a *marking* and consists of a distribution of *tokens* (black dots) on places. When *all* the input places of transitions hold at least one token each, the transition is said to be *enabled* (it may fire). After a transition fires, it places tokens on *all* its output places, thereby enabling other transitions.

A Petri Net can be analyzed by constructing a *reachability graph* (also called computation flow graph or reachability tree). This graph consists of all the states (markings) which can be reached from the initial state (initial marking) by any sequence of transition firings. Reachability graphs are often very large, and are sometimes infinite. However, for many interesting systems, the graph is finite and even small. In such cases, deadlocked states (those with no successors) and critical transitions between states (transitions which eventually lead to deadlock) can be identified [Razouk R. 80]. It should be noted that the reachability graph is independent of any notion of *time:* It yields results about time-independent sequences of events. If the system being modeled has time dependencies, many of the markings in the reachability graph may not, in fact, be reachable. Time-independent reachability analysis can therefore only be used to detect *potential* deadlocks. If the time-independent reachability graph is found to be deadlock-free, we can assume that no time dependencies (short of an infinite delay for a transition) can introduce a deadlock.

Since many interesting concurrent systems have timing dependencies (e.g. real-time systems and communication protocols), it is necessary to introduce the notion of time into Petri Nets. Merlin and Farber [Merlin P 76] extended Petri Nets to include *Min times* and *Max times* associated with transitions. Min times define delays during which transitions must remain enabled before they can fire. Max times define maximum delays before a transition must fire. While a transition is enabled, tokens remain on its input places. This permits other transitions, with shorter delays, to *rob* the transition of its enabling token. This mechanism is particularly useful in modeling timeouts in communication protocols. Merlin and Farber used their Time Petri Nets to design *recoverable* systems (systems which can recover from transient failures).

Ramchandani [Ramchandani C. 74] introduced time by associating simple delays with transitions in a Petri Net. Ramamoorthy and Ho [Ramamoorthy C 80] then used this Extended Timed Petri Net model to analyze system performance. For a restricted class of systems which can be modeled using decision-free nets, Ramamoorthy showed that performance can be analyzed efficiently. Decision-free nets are a very restricted class of nets which involve neither decisions nor non-determinism. In decision-free nets, each place can be connected to the input of no more than *one* transition and to the output of no more than *one* transition. By placing this restriction on the nets, the issue of whether the tokens remain on the input places during the firing delay of a transition becomes moot. Unfortunately, the decision-free restriction is particularly bothersome in modeling communication protocols where decision places are common (See model of transmission medium in figure 1). Ramamoorthy's work also showed that performance analysis of general Petri Nets is NP-complete. This is indeed a discouraging result, but it has not (nor should it) discourage further work in the area, since many cases have been shown to be easily analyzable.

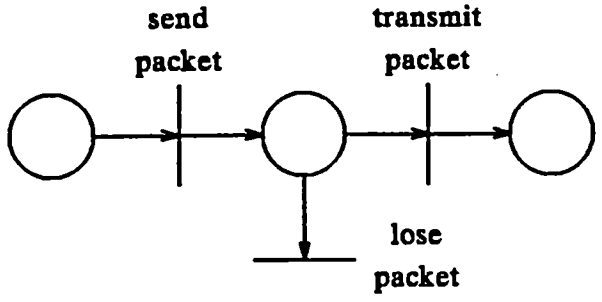

Figure 1. Petri Net of lossy transmission medium

Zuberek [Zuberek W. 80] also used timing delays associated with transitions. The restrictions on Petri Nets were relaxed to permit decisions to be modeled. The Nets were, however, limited to *free-choice* nets. In a free-choice net, only *one* place can be involved in any decision. Zuberek's extensions also required that each transition enabled by a free-choice place be assigned a firing probability. This extension permits the construction of elegant models of lossy transmission media. Zuberek's free-choice limitation remains overly restrictive in modeling communication protocols. Figure 2 shows a typical model of a process waiting for a message with a sequence count of 0 or 1. Depending on the sequence count, the message is processed differently. The resulting net is not free-choice. In fact, if only one message can exist at any time, the net involves no conflicts since at most one of the two transitions will ever be enabled.

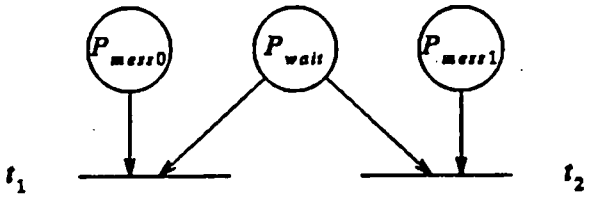

Figure 2. Non-free-choice Petri Net

Zuberek's definition of time assumes that when a transition is enables it immediately *starts firing* by absorbing its input tokens. A transition then *continues to fire* during its defined delay, and then *finishes firing*. This is a subtle point which differs from the Merlin and Farber definition of time. Zuberek also introduced an analysis technique based on a Timed Reachability Graph (which is referred to as a GRID). The Timed Reachability Graph (TRG) differs from a reachability graph in one key aspect: *Time* is introduced as part of the definition of the state of a net. If absolute times are used to describe each state, the reachability graph becomes infinite. Zuberek reduced the state space by using the *remaining firing time* of currently firing transitions as part of the state component. Therefore, a state in the TRG consists of:

1. a marking

2. a vector of remaining firing times (one for each transition)

A state containing a non-zero remaining firing time (RFT) indicates that the transitions in question are firing while the modeled system is in that state. The TRG can be constructed by systematically calculating the successors of each state, starting from the initial state. For a given state, a successor is reached as the result of a transition beginning to fire or, if there are no enabled transitions, as the result of a transition finishing firing (thereby changing the marking and possibly enabling some transitions). In the later case, time must elapse. This is

accomplished by reducing the remaining firing time of currently firing transitions until one or more transitions finish firing.

This is a simple procedure which is essentially identical to the basic algorithm for a Timed Petri Net discrete event simulator. The only difference is that a simulator builds *one* successor state while an analyzer must build *all* successor states. Multiple successors only exist when a free-choice place holds a token. In such cases the analyzer must determine all the possible sets of transition which can be fired simultaneously. Zuberek calls these sets *selectors*. The probability associated with each selector is the product of the probabilities of the free-choice transitions in each *selector*.

Figure 3a shows a free-choice Timed Petri Net. Each transition t has a firing time t_f. Figure 3b shows the Timed Reachability Graph while figure 3c shows the description of each state, including the current marking and the remaining firing times of the transitions. For a comparison, figure 3d shows the standard reachability graph of the un-timed Petri Net. While the timed version contains more states in this example, this is not always the case. In the example in figure 3, probabilities assigned to transitions t_1 and t_3 are used to determine the probabilities associated with each edge out of states 1 and 7 in the timed reachability graph. These probabilities, combined with using time delays as edge weights, can be used to calculate cycle times and to derive resource utilization measures.

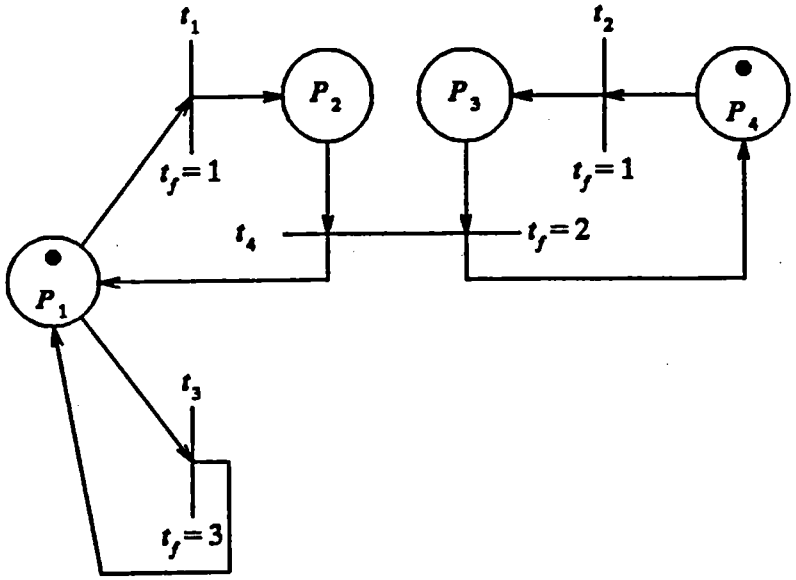

Figure 3a. Example of free-choice net

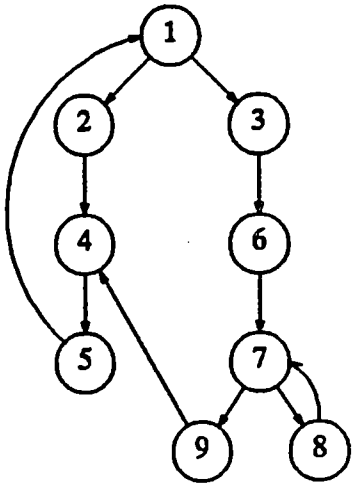

Figure 3b. Timed Reachability Graph

State	Marking				RFT			
	p_1	p_2	p_3	p_4	t_1	t_2	t_3	t_4
1	1	0	0	1	0	0	0	0
2	0	0	0	0	1	1	0	0
3	0	0	0	0	0	1	3	0
4	0	1	1	0	0	0	0	0
5	0	0	0	0	0	0	0	2
6	0	0	1	0	0	0	2	0
7	1	0	1	0	0	0	0	0
8	0	0	1	0	0	0	3	0
9	0	0	1	0	1	0	0	0

Figure 3c. Descriptions of Reachable States

The work discussed above deals with deterministic times associated with transitions. Molloy [Molloy M. 81] has shown that by assuming exponential distributions of delays, Markov Chain analysis can be used to obtain performance measures. The analysis requires that the un-timed reachability graph be constructed. While this work is particularly well suited for modeling systems at a high level of abstraction, it cannot handle fixed timing constraints. Molloy's analysis is based on constructing the un-timed reachability graph and is therefore difficult to use when the graph is large or infinite. In the case of infinite graphs (such as those resulting from models of timeouts), the Petri Net model must be artificially altered to guarantee the reachability graph is finite.

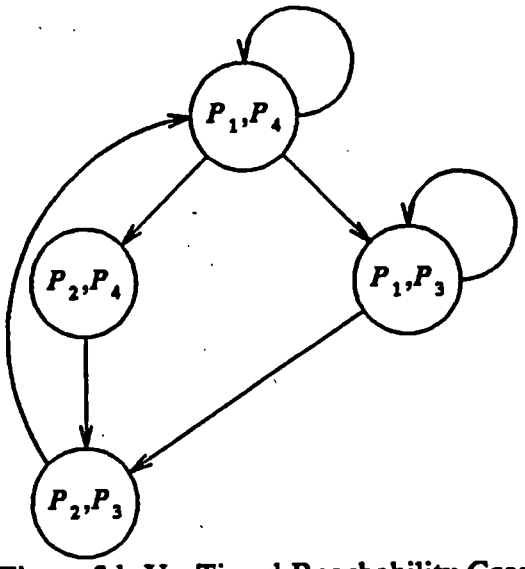

Figure 3d. Un-Timed Reachability Graph

Section 2 of the paper presents a version of Timed Petri-Nets which combines Merlin's Min/Max times with Zuberek's transition delays. Zuberek's restriction regarding free-choice nets is relaxed using the concept of *conflict sets*. Section 3 then shows how Zuberek's Timed Reachability Graph can be constructed using the more general model. A procedure for analyzing the graph is then used to derive performance measures.

2. Yet another version of Timed Petri Nets

Our version of Timed Petri Nets relies on two concepts: 1) enabling times and firing times, and 2) conflict sets

2.1 Enabling times and firing times

Each transition in the net must remain enabled for a time period t_e (its enabling time) before it can fire. A transition is then said to be *firable*, and immediately *begins* firing by absorbing tokens from its input places. The transition *continues* to fire for a period t_f (its firing time). The transition then *finishes* firing and places tokens on its output places. Enabling times are exactly Merlin's Min times. Firing times are exactly Zuberek's transition delays. Combining the two solves several problems:

1. Timeouts can be modeled elegantly. A timeout transition is one with a non-zero enabling time. Although the transition is enabled, it cannot fire. The counting of enabling time starts from the point when a transition first becomes enabled. For the sake of simplicity, we assume that the transition cannot be enabled a second time. This restriction can be achieved by requiring the net to be *safe* (no place can ever hold more

than one token).

2. Transitions cannot be dissabled after they start firing. In our opinion, this is a more accurate model of timing delays. In Merlin's definition of time, a transition can be disabled at any time during its period of activity. Merlin's approach has the advantage of modeling ranges of time delays. However, our model can also be extended to describe ranges of delays while retaining the assumption that transitions cannot be terminated once they start firing. For the purpose of this paper, we assume fixed delays to simplify the analysis.

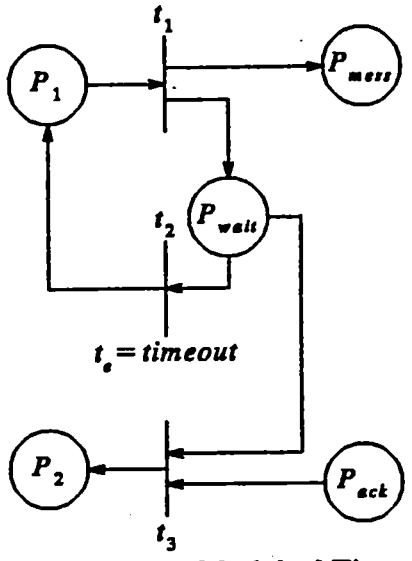

Figure 4. Model of Timeout

Figure 4 shows an example of using enabling times to model timeouts. Transition t_1 models a transmitter sending a message (place P_{mess}) and waiting for an acknowledgment (place P_{wait}). Transition t_2 has an enable time equal to the timeout period, while transition t_3 has a zero enabling time. As soon as an acknowledgment is received (place P_{ack}), the token is removed from place P_{wait}, thereby disabling the timeout. The issue of which transition fires if the acknowledgment arrives *exactly* when the enable time has elapsed, is discussed in section 3.

2.2 Conflict Sets

Zuberek's analysis was restricted to free-choice nets in order to simplify the derivation of branching probabilities associated with edges in the reachability graph. By limiting the analysis to free-choice nets, every node in the reachability graph with more than one outgoing arc (decision vertex) corresponds to one or more free-choice place. If only one free-choice place is involved in the decision, then the probabilities assigned to its output transitions can be

used directly to label the edges of the graph. There is never any concern that an output transition of a free-choice place may not be enabled. Before we can relax the free-choice restriction to allow for modeling of communication protocols, a method must be defined to derive the branching probabilities.

In this work, we partially relax the free-choice restriction. We allow places other than the decision place, to be involved in the firing of a decision transition. We retain the restriction that all conflicting transitions must be mutually disabling, e.g. firing one is guaranteed to disable all the others. This restriction cannot be easily formulated as a restriction on the structure of the net. Whether a net meets this condition or not must be determined during the analysis. It should be noted that free-choice places and their output transitions always meet the above condition.

We refer to each set of potentially conflicting transitions as a *conflict set*. More formally, every transition t_i belongs to exactly one conflict set C such that:

$$C = \{ t_i \mid Inp(t_i) \cap Inp(t_j) \neq \varnothing \}, \text{ where } Inp(t) \text{ is the set of input places of transition } t$$

The above definition implies that conflict sets cannot overlap. With each transition in a conflict set, the user must define a relative firing frequency f. A firing frequency of zero indicates that other transitions, if firable, always have priority. When a decision vertex is reached in constructing the TRG, the probability of firing a firable transition t_i, belonging to a conflict set C, is calculated as follows:

$$\frac{f_i}{\sum_{j \mid t_j \in C \wedge t_j \text{ is firable}} f_j}$$

If only *one* transition is firable, then the probability of firing it is 1, regardless of firing frequency. If all the transitions in a conflict set are firable, then the firing frequencies are the branching probabilities.

Conflict sets are illustrated in figure 5. The conflict set $\{t_1: 0.8, t_2: 0.1, t_3: 0.1\}$ indicates that when all transitions are firable, t_1 should fire 8 times as frequently as t_2 or t_3. If, in some state, t_3 is not firable but t_1 and t_2 are, the ratio remains constant, yielding a probability of 0.888 that t_1 will fire.

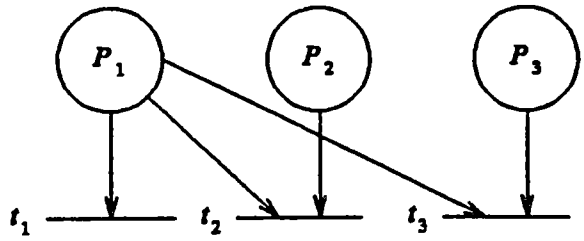

Figure 5. Example of Conflicting Transitions

3. Performance Analysis Using Timed Reachability Graphs

In order to derive performance measures, timed reachability graphs must be built and analyzed. In the next two sections we discuss how Zuberek's Timed Reachability Graphs (TRG) can be extended to include enabling times, and how they can be systematically analyzed to derive performance measures.

3.1 Timed Reachability Graph

In order to incorporate enabling times into TRG's, we must add an additional state component. A state must now include a vector of *remaining enable times (RET)*. As soon as a transition becomes enabled, the corresponding RET must be set to the transition's enable time (t_e). In calculating successor states it is no longer sufficient to determine enabled transitions. In order to fire, a transition must be *firable*. A transition is firable if, and only if, it is enabled and its RET is zero. If an enabled transition is disabled by the firing of another transition, its RET must be reset to zero. If no transitions are firable, both the RET's and RFT's must be decremented until an enabled transition becomes firable and/or a transition finishes firing. Figure 6 shows a more detailed procedure for constructing the Timed Reachability Graph. This procedure is illustrated in the example below.

Figure 7 shows a portion of a safe Timed Petri Net. Place P1 is a free-choice place and enables transitions t_3 and t_4. The resulting conflict set is $\{t_3: 0.1, t_4: 0.9\}$. Place P2 is *not* a free-choice place and can enable transitions t_1 and t_2. Transition t_1 actually models a timeout. The resulting a conflict set is $\{t_1: 0, t_2: 1\}$. By associating a zero probability with t_1 we ensure that if, at some point, both transitions are firable, that t_2 has priority over t_1. If, however, t_1 alone is firable, then it will fire. Since t_1 is the only timeout transition, the enable times for transitions t_2 - t_5 are zero. We assume that the firing times for t_1 - t_5 are <1,2,2,4,4>.

Starting from the marking shown in figure 7, the initial state of the TRG is
S_0 = { marking = <1,1,0,0,0,1,0>, RET = <2,0,0,0,0>, RFT = <0,0,0,0,0> }

At this point, transitions t_1, t_3, t_4 and t_5 are enabled, but only transitions t_3, t_4 and t_5 are fir-

```
Given State S
Let F be the set of firable transitions
if F = ∅
    Let Tmin = smallest non-zero RET or RFT in S
    Generate state S' from S by subtracting Tmin from all
        non-zero RET and RFT in S
    For all transitions t_i whose RFT > 0 in S and RFT = 0 in S'
        add tokens to output places of t_i
    For all transitions t_j which become enabled in S'
        set RET of t_j to t_ej in S'
    Assign Tmin as the time delay for the edge between S and S'
Else
    Partition F into firable conflict sets
    Let the set of selectors Sel = cross product of firable conflict sets
    Calculate the probability of using each selector s in Sel
    For every selector s in Sel
      generate a successor state S' from S
        Remove tokens from input places of transitions in s.
        Set the RFT of each transition in s.
        For every transition which becomes disabled, in S' reset its RET to 0.
        Assign a zero time delay to the edge from S to S'
endif
```

Figure 6. Procedure for generating TRG

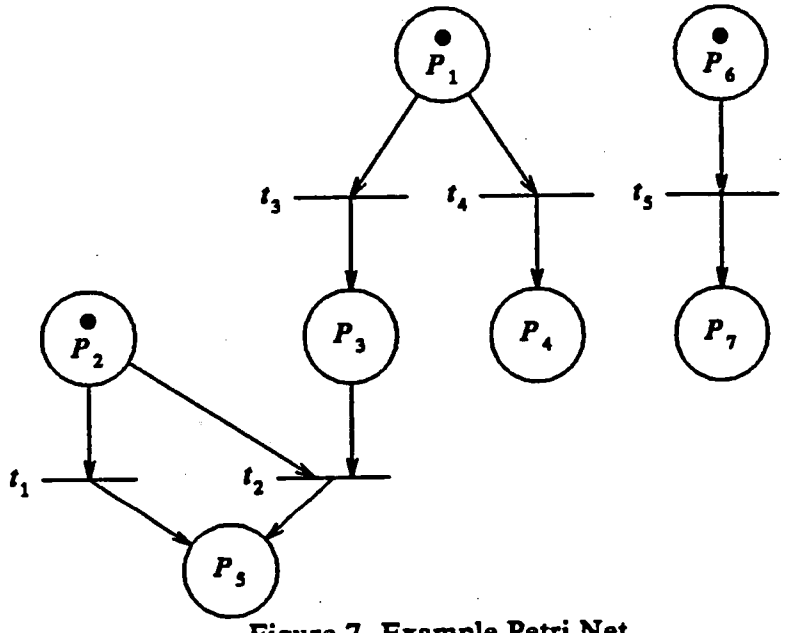

Figure 7. Example Petri Net

able. We partition these firable transitions into *firable conflict sets* : $\{t_3, t_4\}$, $\{t_5\}$. These firable conflict sets indicate that t_3 and t_4 cannot both fire since firing one disables the other. The cross product of these two sets yields the set of *selectors* to be used in constructing successor states.

$$Sel = \{ <t_3, t_5>, <t_4, t_5>\}$$

The probabilities associated with the two selectors are 0.1 and 0.9 respectively. Selector $<t_3$,

t_5> yields the successor state

S_1 = { marking = <0,1,0,0,0,0,0>, RET = <2,0,0,0,0>, RFT = <0,0,2,0,4> }

Selector <t_4, t_5> yields the successor state

S_2 = { marking = <0,1,0,0,0,0,0>, RET = <2,0,0,0,0>, RFT = <0,0,0,4,4> }

Since there are no other firable transitions, successors of S_1 and S_2 are obtained by decrementing the RET and RFT vectors by 2. In state S_1 the result is that t_3 terminates, and that both t_1 and t_2 become firable (but t_2 will fire with probability 1). In state S_2, the result is that t_2 becomes firable, and that both t_4 and t_5 continue firing.

The above procedure is easy to automate and is the basis of an analysis tools currently being developed. The procedure will also be used as the basis for a TPN simulator.

3.2 Analysis of Timed Reachability Graphs

Zuberek's work demonstrated a method of analyzing Timed Reachability Graphs which yields useful resource utilization measures. If an assumption is made that each transition in the net corresponds to some activity of interest, or requires the use of some resource, then resource utilization measures can be obtained by calculating the fraction of the total time that at least one transition of interest is firing. In [Zuberek W. 80], Zuberek presented two examples of reachability graphs derived from nets that model simplified computer architectures. The graphs were used to calculate average instruction times and average memory unit utilization. The method Zuberek presented can be applied directly to the more general Timed Reachability Graph presented above. Since Zuberek did not present a general procedure for deriving the utilization measures, such a procedure is discussed below. The algorithm is shown in figure 8. In the discussion, we assume that the system to be analyzed is well behaved. The issues of incorrect behavior (e.g. deadlocks) are beyond the scope of this paper.

Construct the decision graph from the reachability graph

Since we are considering only cyclic nets, every vertex in a reachability graph has either one edge leading from it or more than one edge leading from it. Vertices that fall into the latter category are refered to as *decision vertices*. Each edge leading away from a decision vertex defines a single path to another decision vertex. That is, if we traverse an edge e from a decision vertex, and continue traversing edges until the next decision vertex is reached, we have no choice as to which edges are traversed. The idea of a *decision graph* is that all the intermediate vertices between the decision vertices are ignored, and the edges between decision vertices are coalesced into one edge.

Construct the decision graph and calculate for
each edge e_i in the graph a resource busy time b_i
Compute relative edge traversal frequencies r_i for each edge.
Obtain weighted time $w_i = r_i t_i$
Compute the fraction of the total time spent on each edge.

$$p_i = \frac{w_i}{\sum_{all\ j} w_j}$$

Compute fraction of total time during which resource is used.

$$resource\ utilization = \sum_{all\ i} p_i b_i$$

Figure 8. Algorithm for analyzing TRG.

More precisely, to construct a decision graph, start with a set of vertices $\{D_1, D_2, \ldots D_n\}$ which are in one-to-one correspondence with the set of decision vertices $\{R_1, R_2, \ldots R_n\}$ in the reachability graph. Each R_i has a set $\{e_{i1}, e_{i2}, \ldots e_{im}\}$ of edges leading away from it. For every e_{ik} for every R_i, follow edges leading away from R_i, starting with e_{ik}, until another decision vertex R_j is reached (Possibly i = j). Add an edge (D_i, D_j) to the decision graph, and assign a time equal to the sum of the times of the edges traversed, and a probability equal to the probability of e_{ik}. These quantities will be called t_i and pr_i, respectively. Since paths between decision vertices may converge, a more efficient implementation would be to traverse edges in reverse order, summing times along the way, and recursively computing new paths when convergence points are encountered. This would save the cost of traversing the common edges and summing their times more than once.

Another quantity necessary to measure performance is the proportion of time on each edge of the decision graph during which a particular resource is used, or an activity of interest occurs (i.e. the fraction of time during which at least one of a set of designated transitions is firing). The fraction for edge e_i will be called b_i (for busy). These values can be obtained by marking edges on the TRG during which the resource is used, and then computing the b_i's as the decision graph is constructed.

Since we are interested in cyclic processes, every vertex in the decision graph should be reachable from every other vertex; i.e., the vertices should form a knot. Possible anomalies include vertices that are not part of a knot, and more than one knot. If an anomaly exits, it indicates either an error in the net, or that the behavior of the process being modeled is not appropriate for this technique. We shall not discuss methods of detecting whether or not a single knot which includes every vertex in the graph exists. It should be noted that if there is no anomaly, then the method described for constructing the decision graph automatically

disregards vertices in the reachability graph that correspond to initial start-up states.

Below is a summary of the notation introduced in this section. All values are associated with the decision graph.

t_i: time to traverse edge e_i

pr_i: probability for edge e_i

b_i: fraction of time on edge e_i that resource is used

Compute relative edge traversal frequencies

In order to compute resource utilization we must find out the fraction of the total time during which the resource is being utilized. To do this, we must find out the fraction of time spent traversing each edge of the decision graph, if we traverse edges endlessly. Since a decision as to which edge to choose has no relation to the time delays, we first calculate relative edge traversals frequencies. The relative number of edge traversals is related to the branching probabilities. Given a decision vertex i, the relative number of traversals of a particular outgoing edge e_{it} is a fraction of the total number of times the incoming edges are traversed.

These relationships can be described as a set of equations, one for each edge. The equations can be constructed as follows:

For each edge e_i do

Set up equations $r_i = pr_i \sum_{j \in I(i)} r_j$

where I(i) = {j | e_i can be traversed immediately after e_j}.

Compute fraction of time spent on each edge

Given the relative number of edge traversals we can compute the relative amount of time spent traversing each edge: $w_i = r_i t_i$. To derive the fraction of time p_i spent traversing each edge e_i:

For all i do

$$p_i = \frac{w_i}{\sum_{all\ j} w_j} \qquad 1$$

Compute the final result

Given the fraction of time spent traversing each edge, we can derive resource utilization measures as follows:

$$final\ result = \sum_{all\ i} p_i b_i \qquad 2$$

4. Analysis of Performance of Communication Protocol

In this section we demonstrate the modeling approach and the analysis technique using a simple communication protocol. The protocol in question is modeled in figure 9. The sender sends a packet (transition t_2) and waits for an acknowledgement. A timeout (transition t_3) is used to recover from lost packets. The receiver waits for a message and sends an acknowledgement immediately (transition t_6). The medium can lose packets (transition t_4) and acknowledgements (transition t_9). In its design, this protocol assumes that timeouts are only triggered if the packet or its acknowledgement have been lost. We assume that the receiver can detect a duplicate message, but that the sender cannot detect a duplicate acknowledgement. This is a trivial protocol, which can be easily extended to be more robust by using alternating bits for message and acknowledgement sequencing. For the sake of brevity, we have opted for the simpler, less robust, protocol.

Our objective in the performance analysis is to calculate the utilization of the sending channel and the effective throughput of the protocol. We assume the transmission medium loses 5% of transmitted packets and acknowledgements. In order to validate the analysis techniques, we have chosen values for delays which are consistent with those chosen by Molloy in [Molloy M. 81]. We are therefore assuming a 9600 baud link, transmitting 1024-bit packets. We assume that acknowledgment packets are also 1024 bits long. In order to calculate peak throughput, we assume that packets are made available for transmission as soon as the previous packets was successfully sent. We only introduce a minor delay to model overhead (1 millisecond). We assume a 1 second timeout and a 13.5 millisecond delay for processing of a received message or acknowledgment. Figure 10 shows how the above measures translate into enabling and firing times of the transition in the net.

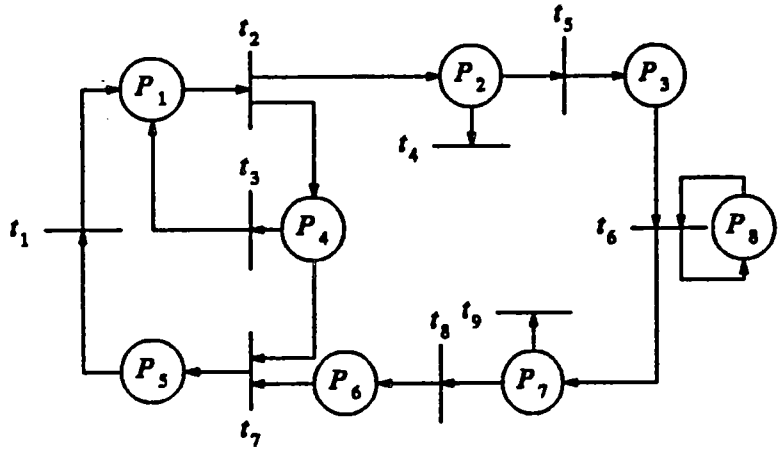

Figure 9. Model of Simple Protocol

Transition	Enable Time (milliseconds)	Firing Time (milliseconds)
t_1	0	1
t_2	0	1
t_3	1000	1
t_4	0	106.7
t_5	0	106.7
t_6	0	13.5
t_7	0	13.5
t_8	0	106.7
t_9	0	106.7

Figure 10. Enabling and Firing Times

The net contains 3 conflict sets containing more than one transition: $\{t_4: 0.05, t_5: 0.95\}$, $\{t_3: 0, t_7: 1\}$, and $\{t_8: 0.95, t_9: 0.05\}$.

The Timed Reachability Graph for this net, starting from the initial marking $< P_1, P_8 >$, is shown in figure 11. The description of each state is shown in figure 12. The Decision Graph is shown in figure 13. The values for the b_i's are derived by calculating the fraction of each edge during which either transition t_4 or t_5 are firing. From the Decision Graph, we can derive the following equations for the relative number of edge traversals:

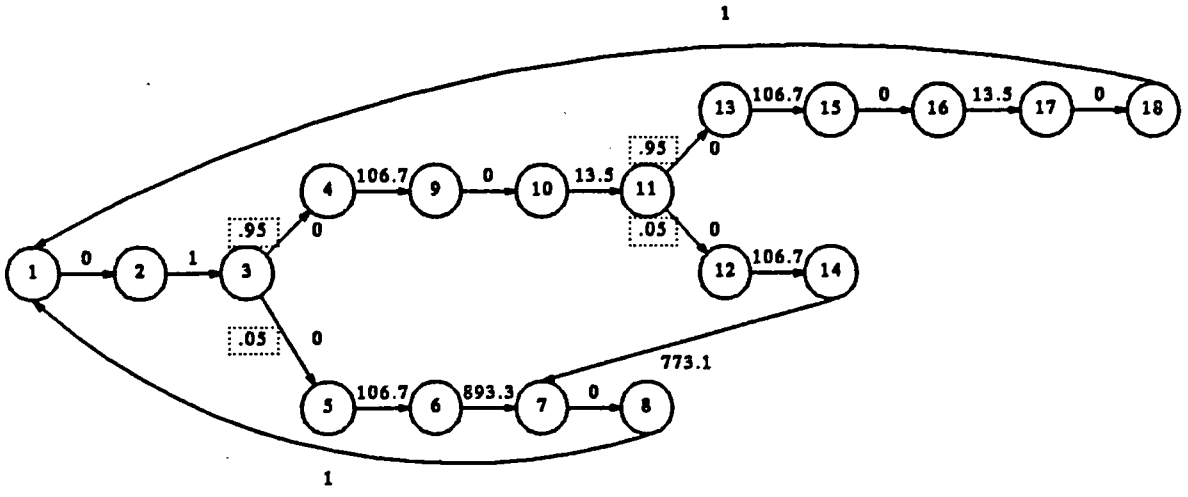

Figure 11. Timed Reachability Graph of Simple Protocol

State	Marking								RET	RFT								
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	t_3	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	1	1000	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	1	1000	0	0	0	0	106.7	0	0	0	0
5	0	0	0	1	0	0	0	1	1000	0	0	0	106.7	0	0	0	0	0
6	0	0	0	1	0	0	0	1	893.3	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
9	0	0	1	1	0	0	0	1	893.3	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	893.3	0	0	0	0	0	13.5	0	0	0
11	0	0	0	1	0	0	1	1	879.8	0	0	0	0	0	0	0	0	0
12	0	0	0	1	0	0	0	1	879.8	0	0	0	0	0	0	0	0	106.7
13	0	0	0	1	0	0	0	1	879.8	0	0	0	0	0	0	0	106.7	0
14	0	0	0	1	0	0	0	1	773.1	0	0	0	0	0	0	0	0	0
15	0	0	0	1	0	1	0	1	773.1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	13.5	0	0
17	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Figure 12. Description of State in TRG

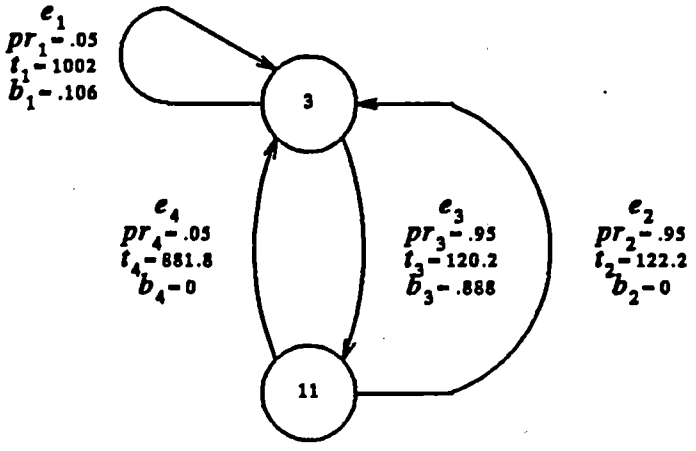

Figure 13. Decision Graph

$$r_1 = .05(r_2 + r_4), \quad r_2 = .95(r_3)$$

$$r_3 = .95(r_2 + r_4), \quad r_4 = .05(r_3)$$

We then solve for the r's by setting r_1 to 1. The resulting values are: $r_1 = 1, r_2 = 19, r_3 = 20, r_4 = 1$. Using equation 1, we calculate the fraction of time spent on each edge to be:

$$p_1 = \frac{1002}{6610} = .152, \quad p_2 = \frac{2322}{6610} = .351$$

$$p_3 = \frac{2404}{6610} = .364, \quad p_1 = \frac{882}{6610} = .133$$

Using equation 2 we calculate the utilization of the medium to be .339.

In order to calculate the effective throughput of the protocol we note that one the average it takes 6.610 seconds (the total time in the graph) to transmit 19 messages (traverse edge e_2 r_2 times). The throughput is therefore 2.87 messages per second. This compares to the maximum throughput of 2.75 messages per second calculated by Molloy [Molloy M. 81].

Conclusions

This paper has presented a new extension of the basic Petri Net model. The introduction of enabling times and firing times allows designers to accurately model timeouts and processing delays. Limitations previously placed on an analysis technique have been relaxed, thereby allowing interesting models of communication protocols to be constructed. Algorithms for constructing and analyzing timed reachability graphs were presented. The algorithms can be easily automated and are expected to be useful in simulating as well as analyzing systems modeled by Timed Petri Nets.

Some restriction on the Petri Net models remain. We have assumed that nets are safe. We conjecture that this restriction can be relaxed. Methods for calculating branching probabilities and for calculating enable times for unsafe nets are currently being investigated. We are also currently investigating the relationship between verifying the correctness of protocols and evaluating their performance.

References

[Berthelot G. 82] Berthelot, G. and Richard Terrat, "Petri Net Theory for the Correctness of Protocols," *Protocol Specification, Testing and Verification*, North Holland Pub. Co., (1982).

[Merlin P. 76] Merlin, P. and D. Farber, "A Methodology for the Design and Implementation of Communications Protocols," *IEEE Transactions on Communications*, COM-24, 6 (June 1976).

[Molloy M. 81] Molloy, M. "On the Integration of Delay and Throughput Measures in Distributed Processing Models", Computer Science Dept., University of California, Los Angeles, Report No. CSD-810921, September 1981.

[Peterson J. 81] Peterson, J.L. *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs, New Jersey, Prentice Hall, Inc. 1981.

[Petri C. 65] Petri, C. A. "Kommunikation mit Automaten," *Schriften des Rheinisch-Westfalischen Institutes fur Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany 1962; translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Grifiss Air Force Base, N.Y., 1965.

[Ramamoorthy C. 80] Ramamoorthy C.V. and G.S. Ho, "Performance Evaluation of Asynchronous Concurrency Systems using Petri Nets," *IEEE Transaction on Software Engineering*, SE-6, 5 (September 1980), 440-449.

[Ramchandani C. 74] Ramchandani, C. "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Ph.D. Thesis, MIT 1974, Project Mac Report No. MAC-TR-120.

[Razouk R. 79] Razouk, R.R., M. Vernon and G. Estrin "Evaluation Methods in SARA - the Graph Model Simulator" *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, August 1979, pp. 189-206.

[Razouk R. 80] Razouk, R.R., and G. Estrin "Modeling and Verification of Communication Protocols: The X.21 Interface" *IEEE Trans. on Computers*, Vol C-29, No. 12, December 1980, pp. 1038-1052.

[Sifakis J. 77] Sifakis, J. "Petri Nets for Performance Evaluation," *Measuring, Modelling and Evaluating Computer Systems*, Proceedings of the 3rd International Symposium, IFIP Working Group 7.3, H. Beilner and E. Gelenbe (eds.), North-Holland Pub. Co. 1977, pp. 75-93.

[Symons F. 80] Symons, F.J.W., "Verification of Communication Protocols using Numerical Petri Nets," *Australian Telecommunication Research*, 14,1 (1980) 34-38.

[Vernon M. 83] Vernon, M.K., E. de Souza e Silva, and G. Estrin "Performance Evaluation of Asynchronous Concurrent Systems: The UCLA Graph Model of Behavior" *9th International Symposium on Computer Performance Modelling, Measurement and Evaluation*, College Park, Maryland, May 25-27, 1983.

[Zuberek W. 80] Zuberek, W.M., "Timed Petri Nets and Preliminary Performance Evaluation," *7th Annual Symposium on Computer Architecture*, 1980, pp. 88-96.