On Petri Nets and Self-Stabilization of Communication Protocols

Wuxu Peng*
Department of Computer Science
Southwest Texas State University
San Marcos, TX 78666

Abstract

We discuss, using Petri nets as the concurrent model, the issue of self-stabilizing communication protocols. We argue that self-stabilizing extensions of communication protocols should not interfere with their normal operations by giving a new definition of self-stabilizing extension *and a ring net that meets the new definition. We argue that communication protocols are usually delay sensitive and* time-Petri nets *is a more suitable formal model (than ordinary Petri nets) to express self-stabilizing property. We also explore the possibility and suitability of using Petri nets with a new type of transitions – the* max transitions *– to express self-stabilizing communication protocols. Finally, to facilitate practical implementations, two types of interferences –* external *and* internal *interferences – are formally identified and discussed.*

1 Introduction

Communication protocols are an integrated and critical part of distributed systems and computer networks. Concisely specifying and efficiently verifying communication protocols are always major challenges to network designers.

The traditional approach to verifying communication protocols is similar to the well-known *weakest pre-condition* and *strongest post-condition* technique employed in verifying sequential programs. For example, to verify that a protocol is free of deadlocks, we may show that if the protocol starts from a legal state then it can never be in an illegal deadlock state, i.e. a state in which several machines are waiting for messages from one another in order to proceed. This traditional approach has severe problems, however. The distributed nature of distributed systems and computer networks makes such systems vulnerable to events absent in centralized systems. For instance, the communication channels linking machines may break or suffer from outside interference. Such a problem may cause either loss or corruption of messages. A corrupted message that passes the normal error detection mechanism may cause the system to enter an illegal state. A power surge in one machine may cause this machine to mal-function transiently and hence send conflict messages to different machines in the system. We observe that in both these two cases, a little perturbation can cause a system to stray from its legal states and remain there thereafter. To bring the system back to its legal states, drastic measures such as shutting down and rebooting the system have to be taken.

The concept of self-stabilization of computer systems was first proposed by E. W. Dijkstra in his seminar paper [5]. Informally, we say that a computer system is *self-stabilizing* if and only if the following two conditions are satisfied:

(1) if the system is in a legal state, it will remain in the set of legal states thereafter provided there is no perturbation;

(2) if some perturbation causes the system to enter an illegal state, then the system can return to a legal state within a *finite* amount of time *without* outside interference.

From the definition, it is clear that a self-stabilizing system is much more robust than a system that is merely logically correct. Self-stabilizing systems are *faults-resilient* in the sense that they can recover from system faults and failures by themselves. For example, when a computer system is rebooted, it is very important to ensure that all system parameters are properly initialized. For a self-stabilizing system, however, we can just turn on the power without concern about its initial state. The self-stabilization property guarantees that the system will eventually enter a legal state within a finite amount of time and remain there thereafter.

*Supported in part by a faculty research enhancement grant from SW Texas State Univ.

0-8186-3670-X/93 $03.00 © 1993 IEEE

Although the concept of self-stabilization was proposed nearly twenty years ago, it did not receive its deserved attention until the middle of 1980s. After Dijkstra's paper [5], Kruijer in [13] considered the issue of self-stabilization in tree-structured systems with distributed control. Bastani, Yen and Chen considered the self-stabilization problem of a class of distributed programs for industrial process-control systems in [1]. Brown, Gouda, and Wu considered the self-stabilization problem for token ring systems in [2]. The behavior of each machine in the ring is characterized by a regular expression and three *non-interfering* and *delay-insensitive* self-stabilizing protocols were presented. Gouda and Multari examined the issue of designing self-stabilizing communication protocols in [8]. The most important conclusion from that paper is the observation that the existence of both timeout and infinite number of states is a necessary condition for a communication protocol to be self-stabilizing. Burns and Pachl showed in [3] a *uniform* self-stabilizing solution (i.e. each machine in the ring executes the same code) for unidirectional rings where the number of machines is *prime*.

Research on the issue of self-stabilization of Petri nets only started in the late of 1980s. Cherkasova, Howell, and Rosier considered the issue of self-stabilizing bounded Petri nets in [4]. Their definition of self-stabilization is based on reachability sets. Ghosh discussed in [7] the issue of self-stabilizing a special class of Petri net models – the marked graphs. His definition is based on sequences of firing transitions and self-stabilization is achieved by using inhibitor arcs.

In this paper we consider the issue of self-stabilization of communication protocols expressed in Petri nets. We first argue that self-stabilizing extension of any communication protocol should not interfere with the normal operation of the protocol. This claim is supported by a new definition of *self-stabilizing extension* and illustrated by a new self-stabilizing extension of a ring net from [7].

Timeout is a fundamental technique used in communication protocols. For instance, in a token ring, timeout is used to detect loss of tokens. In sliding window protocols, timeout is used to retransmit previously sent but not yet acknowledged messages. As pointed out in [8], timeout is a necessary condition for communication protocols to achieve self-stabilization. The need of timeout in communication protocols can be attributed to the distributed nature of distributed systems and computer networks. Unreliable communication channels may lose or corrupt messages. Individual machines can crash due to various hardware and/or software failures. The need of timeout indicates that communication protocols are delay sensitive, not delay insensitive. For example, long delays in communications are normally treated as errors because most users usually cannot tolerate an excessively long delay. Therefore delay insensitive protocols, such as the three elegant protocols in [2], may not be suitable for many practical applications.

One elegant model for delay sensitive communication protocols is *Time-Petri nets* (TPNs), introduced in [14]. Informally, a TPN is just an ordinary Petri net with additional restrictions on the timing of firing of transitions. We argue that TPNs are more suitable (than Petri nets) to express self-stabilizing property of communication protocols. We show that a time-sensitive token ring can be extended to be self-stabilizing using a TPN that is operating at half of the speed of the original net. The convergence rate (the rate at which a disturbed system returns to a legal state) of the extended net can be accurately bounded.

We also explore the possibility of using *max transitions* to express self-stabilizing communication protocols. The notion of max transitions was originally proposed by Ghodsi and Kant in [6] to model the abort semantics in performance measurement. We show that, with a modified definition, max transitions can often give a more compact representation of self-stabilizing protocols. Two examples, one is the ring net and another is the sliding window protocol, are given to illustrate the point.

We strongly believe that practical implementation consideration should be a critical factor in evaluating possible self-stabilizing extensions of communication protocols. A self-stabilizing extension, no matter how elegant theoretically, has little value if it could not be practically implemented. Out of this consideration, we formally identify two types of interferences – *external* and *internal* interferences – which should be avoided in self-stabilizing protocols.

The remainder of this paper is organized as follows. Section 2 provides necessary definitions and concepts. In particular, we define the concept of self-stabilizing extension of a Petri net. We believe that our definition is innovative. Section 3 Illustrates our self-stabilizing definition through examples. Section 4 discusses, out of practical considerations, how to construct self-stabilizing extension of communication protocols through *decentralized control*. In Section 5 we discuss, through two examples, how to use max transitions to express self-stabilization. The modeling power of Petri nets with max transitions is briefly discussed. Section 6 formally defines and discusses two types of interferences – *external* and *internal* inter-

ferences. Section 7 provides concluding remarks and future research topics.

2 Basic definitions

In this section we define most of the concepts used in this research, including Petri nets, extended Petri nets, max transitions and generalized Petri nets, self-stabilizing extensions of Petri nets. The definition of TPNs is delayed until Section 4.

Definition 2.1 (Petri nets) A Petri net is a four-tuple (P, T, I, O), where $P = \{p_1, \cdots, p_n\}$ is a finite set of **places**; $T = \{t_1, \cdots, t_m\}$ is a finite set of **transitions** satisfying $P \cap T = \emptyset$; $I : T \rightarrow P^\infty$ is the **input** function; and $O : T \rightarrow P^\infty$ is the **output** function, where P^∞ is the **multinet** over P.

Let **N** denote the set of nonnegative integers. A **marking** μ is an assignment of **tokens** to places of a Petri net, i.e. μ is a function from P to **N**:

$$\mu : P \rightarrow \mathbf{N}.$$

A **marked** Petri net is a five-tuple (P, T, I, O, μ), where $C = (P, T, I, O)$ is a Petri net and μ is a marking of C. A transition t in a marked Petri net with marking μ is **enabled** if $\forall p_i \in I(t)\ \mu(p_i) > 0$. A transition t enabled in a marking μ may **fire** and result in a new marking μ' by first removing one token from each of its input places and then adding one token to each of its output places, i.e. $\mu' = \mu - I(t) + O(t)$.

Definition 2.2 (Extended Petri nets) An extended Petri net is a five-tuple (P, T, I, O, I'), where (P, T, I, O) is a Petri net, and $I' : P \rightarrow T$ is the set of **inhibitor arcs**.

Let (P, T, I, O, I', μ) be a marked extended net. A transition $t \in T$ is enabled (with respect to μ) if $\forall p \in I(t)\ \mu(p) > 0$ and $\forall p \in I'(t)\ \mu(p) = 0$. As in the Petri net case, the firing of t will result in a new marking $\mu' = \mu - I(t) + O(t)$. Notice that if $I' = \emptyset$, then the extended net is an ordinary Petri net.

Definition 2.3 (Generalized Petri nets) A generalized Petri net is a six-tuple (P, T, I, O, I', M, S), where (P, T, I, O, I') is an extended net, $M \subseteq T$ is the set of **max transitions**, and $S : T \rightarrow 2^P$, where $\forall p \in S(t)$ there exists a path leading from place p to transition t, is the **input set** of max transitions. The enabling and firing of non-max transitions are the same as in extended net. A max transition t is enabled in a marking μ if $\exists p \in I(t)\ \mu(p) > 0$. An enabled max transition t may fire by first removing one token from each place in $I(t)$, then removing all tokens from each place in the input set $S(t)$, then adding one token to each place in $O(t)$.

Following the notation used in [6], a max transition t will be denoted by a rectangle with an associated copyright sign @. Notice that our definition here is slightly different from the original definition in [6].

For a marked net with a given initial marking, more than one transition may be enabled and thus it is possible for more than one transition to fire simultaneously. Here we adopt the classic interleaving semantics for firing. The set of all possible firing sequences starting from a given marking forms a formal language over T [15].

Definition 2.4 A net N_2 is a self-stabilizing extension of a net N_1 (with respect to a given marking) if and only if every firing sequence α generated by N_2 from an arbitrary initial marking can be expressed as $\beta\alpha'$ satisfying:

(1) $\beta \in T_2^*$ is finite and $\alpha' \in T_1^*$;

(2) there exists a finite sequence $\beta' \in T_1^*$ such that $\beta'\alpha'$ is a legal firing sequence in N_1.

Two comments are in order here. First, the above definition does not require the original net N_1 to be live or safe. Assume that net N_1 is specified by a predicate R. Then by above definition, net N_2 is a self-stabilizing extension of N_1 as long as starting from any initial marking N_2 will satisfy R after a finite number of firing steps. Second, the definition implicitly requires that the extension should not interfere with the normal operations of the original protocol. This is reflected in the requirement that $\beta'\gamma'$ must be a legal firing sequence in N_1.

Another implication of the above definition is that every terminating net is trivially self-stabilizing extension of itself (taking $\alpha' = \varepsilon$). Therefore in this paper we consider net modeling non-terminating communication protocols.

3 Self-stabilizing without interfering normal operations

Ghosh discussed in [7] the issue of self-stabilizing a special class of Petri net models (ring nets and marked graphs). In both cases, inhibitor arcs are used to achieve self-stabilization. Fig. 1 shows a ring net and its two self-stabilizing extensions.

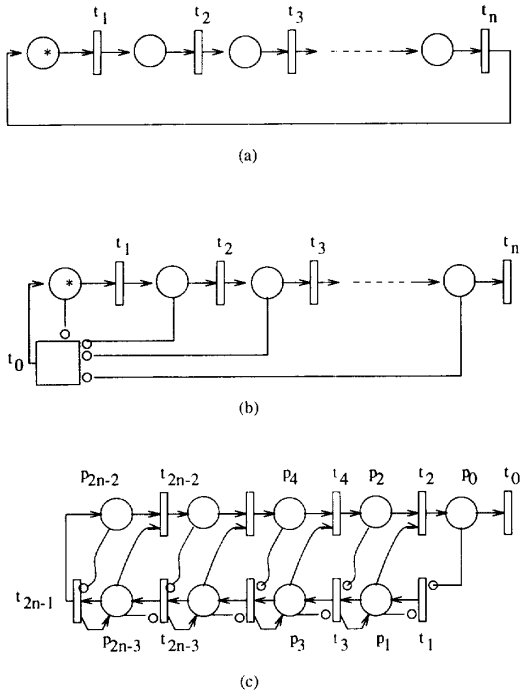

Figure 1: A ring net and its two extensions by Ghosh: (a) the ring net; (b) the first extension; (c) the second extension

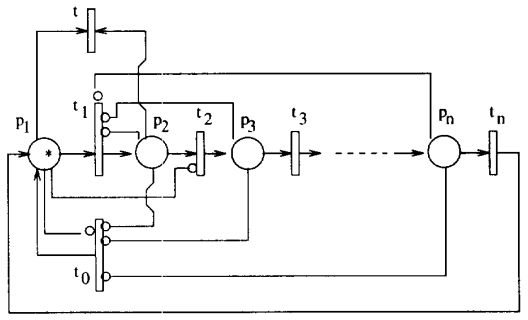

Figure 2: An alternative self-stabilizing extension of the ring net

One problem with the two self-stabilizing extensions is that, in both cases, the system explicitly interferes with the operation of the net, even in the absence of operation errors. For instance, in the first extension on Fig. 1.(b), transition t_n drains all tokens in the net and then the absence of tokens in the whole net causes transition t_0 to be enabled and fire. We believe that in a self-stabilizing extension of any system, the extension should not interfere with the normal operation of the original system and interference will come only after the system enters an illegal state. Incidentally, it is easy to see that both extensions in Fig. 1(b) and Fig. 1(c) do not satisfy Definition 2.4. The net in Fig. 1(a) can be characterized by the regular expression $(t_1 t_2 \cdots t_n)^*$. Every normal firing cycle in both nets in Fig. 1(b) and Fig. 1(c) includes the transition t_0, which is absent in the original net.

A new self-stabilizing extension of the net in Fig 1(a) is given in Fig. 2. As shown by the following theorem, the new extension is guaranteed to operate as usual as long as the system is in legal states and will try to recover from failure as soon as it enters an illegal states.

Theorem 3.1 The network in Fig. 2 is a self-stabilizing extension of the ring net in Fig. 1.(a).

Proof: Self-stabilization is clearly implied by the following three observations:

Observation 1: if there is no token in the net, then transition t_0 can fire and bring the system to a legal state.

Observation 2: assume that there is at one token in place p_1. Transition t_1 cannot fire if there are tokens in any other places. Tokens in places $p_3, p_4, \cdots, p_n$ will eventually reach place p_1.

Observation 3: If there are tokens in both places p_1 and p_2, then only transition t is enabled. Every firing of t will drain two tokens from the net. If place p_2

becomes empty after t's firing but p_1 still holds more than one token, firing of t_1 will move one token from p_1 to p_2. So if initially the total number k of tokens in the net is odd, then after $k/2$ firings of transition t only one token will be left in place p_1. If k is even, then after $k/2$ firings of transition t the net will be empty. □

4 Self-stabilizing through decentralized control

In this section we argue and show that time-Petri Nets, introduced in [14], is more suitable to model the self-stabilizing properties of communication protocols.

The main problem with the ring net in Fig. 2 is the difficulty associated with implementing inhibitor arcs that involve non-neighboring places. For instance, checking if transition t_0 in Fig. 2 is enabled practically asks for the verification of the emptiness of all places in the net, while checking if transition t_1 is enabled demands the verification of the emptiness of places $p_2, p_3, \cdots, p_n$. In a ring net, such a verification is equivalent to a round of communications for consensus among all the transitions, which is is difficult and expensive to implement (considering the fact that such verifications must be performed before every firing of transition t_0 or t_1).

There are at least two approaches that can be used to cope with the above problem. The first approach is to launch the communications checking for absence of tokens periodically, not before every time t_0 or t_1 is fired. Although this may reduce the cost of implementing self-stabilization, it also compromises the self-stabilizing property. The second approach is to make use of the time sensitive property of most practical communication protocols.

Timeout is a fundamental technique used in communication protocols. For instance, in a token ring, timeout is used to detect loss of tokens. In sliding window protocols, timeout is used to retransmit previously sent but not yet acknowledged messages. As pointed out in [8], timeout is a necessary condition for communication protocols to achieve self-stabilization. The need of timeout in communication protocols can be attributed to the distributed nature of distributed systems and computer networks. Communication channels are not reliable and messages can be lost or corrupted. Individual machines can crash due to various hardware and/or software failures.

The need of timeout can also be justified by the fact that long delays in communications are normally

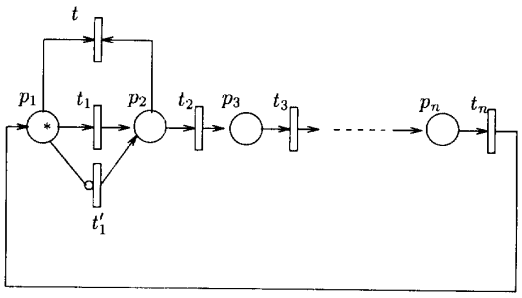

Figure 3: A self-stabilizing extension of the ring net based on TPN

treated as errors because most users usually will not tolerate an excessively long delay. This points out that communication protocols are delay sensitive, not delay insensitive. Therefore delay insensitive protocols, such as the three elegant protocols in [2], may not be suitable for many practical applications.

One elegant model for delay sensitive communication protocols is *Time-Petri nets* (TPNs), introduced in [14]. Informally, a TPN is just an ordinary Petri net with additional restrictions on the timing of firing of transitions. Formally we have the following definition.

Definition 4.1 (cf. [14]) A TPN is a PN in which each transition t_i is associated with two real numbers t_i^* and t_i^{**} satisfying

- $t_i^*, t_i^{**} \geq 0$;
- $t_i^* \leq t_i^{**}$.

The firing rule of a TPN is defined as:

- If t_i has been enabled for a period of time equal to or greater than t_i^*, then t_i **can** fire, using the the same rule for token redistribution as in an ordinary PN.
- If t_i has been enabled for a period of time equal to than t_i^{**}, then t_i **must** fire, using the the same rule for token redistribution as in an ordinary PN.

Intuitively, t_i^* is the minimum time for which transition t_i has been continuously enabled before it can fire, while t_i^{**} is the maximum time for which transition t_i has been enabled continuously before it must fire.

To illustrate the usefulness of TPNs in expressing delay-sensitive communication protocols, we give a self-stabilizing extension of the ring net, as shown in Fig. 3. In that figure, it is assumed that $T = \Sigma_{i=3}^{n} t_i^{**}$,

and the time parameters for transitions in Fig. 3 satisfy the conditions $t_1^* + t_2^* \geq T$, $t_2^* > 0$ and $t^* = t^{**} = 0$, and $t_1'^* \geq T + t_1^{**} + t_2^{**}$,

Theorem 4.1 The network in Fig. 3 is a self-stabilizing extension of the ring net in Fig. 1.(a).

Proof: If the net is empty, then the inhibitor arcs will regenerate tokens.

Now assume that there is more than one token in the net. First we observe that any pair of tokens that are already in place p_1 and p_2 respectively will be removed by the firing of transition t. There are still two cases to consider. **Case 1**: there are two or more tokens in places $p_3, p_4, \cdots, p_n$ at some time point. Due to the requirement that $t_1^* + t_2^* \geq T$, one of these tokens must have been in either place p_1 or p_2 when the other tokens arrive at place p_1. Therefore the transition t will be enabled after at most T time units and the firing of t will remove two tokens from the net. **Case 2**: there are two or more tokens, one is in place p_1 (or p_2) while the others are in places $p_3, p_4, \cdots, p_n$. If the token in p_1 or p_2 is still in p_1 or p_2 when a token from $p_3, \cdots, p_n$ arrives at p_1, these two tokens will enable t and be removed. If the token in p_1 or p_2 is not in p_1 or p_2 when a token from $p_3, \cdots, p_n$ arrives at p_1, the requirement that $t_1^* + t_2^* > T$ ensures that the latter will still be in p_1 or p_2 when the former arrives at p_1. □

The net in Fig. 3 has several distinct properties over the net in Fig. 2. First, the only inhibitor arc from place p_1 to transition t_1' in Fig. 3 connects two neighboring place and transition. This implies that a node need only check the emptiness of its neighboring channel, not all the channels over the net. Second, it shows that self-stabilization of a ring net as shown in Fig. 1(a) can be achieved by operating the net at half of the normal operation speed. Third, the convergence rate of of the net in Fig. 3 can be easily and accurately estimated. Assume that m tokens were in the net at some time point. It can be easily seen, as from the argument in the proof of above theorem, that after at most $(m/2)T$ time units, either all the m tokens are drained from the net or only one token is left. In the late case, the net already returns to a legal state. In the former case, transition t_1' will regenerate a single token after $T' = T + t_2^{**}$ time units, i.e. the net returns to a legal state within $T' + (m/2)T$ time units.

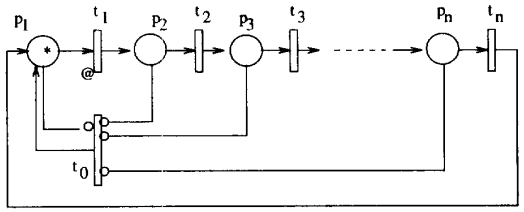

Figure 4: A self-stabilizing extension of the ring net using both max transitions and inhibitor arcs

5 Using max transitions

In last section, inhibitor arcs are used to achieve self-stabilization. As pointed by Ghosh, use of inhibitor arcs seems mandatory in order to achieve self-stabilization. In this section we show that using max transitions together with inhibitor arcs could greatly simplify the representation of self-stabilizing communication protocols.

Fig. 4 shows a net using both max transitions and inhibitor arcs. Transition t_1 is the only max transition with input set $\{p_1, p_2, \cdots, p_n\}$.

Theorem 5.1 The network in Fig. 4 is a self-stabilizing extension of the ring net in Fig. 1.(a).

Proof: Similar to the net in Fig. 2, the case where the net is empty is handled by transition t_0. By the definition of max transitions, the firing of transition t_1 will remove all tokens from all its input places. Hence every firing of t_1 will remove all the tokens in places $p_1, p_2, \cdots, p_n$. □

Our second example, as shown in Fig. 5, is a self-stabilizing extension of the well known sliding window protocol. This example further illustrates the usefulness of max transitions in specifying and expressing self-stabilizing property. A good introduction to sliding window protocols and their self-stabilizing versions can be found in [16, 8]. In Fig. 5, the input sets $S(t_3) = \{p_1, p_2, p_3, p_5\}$, $S(t_{11}) = \{p_7, p_8, p_4, p_6\}$. The possible error where there are tokens in both places p_1 and p_2 (or p_7 and p_8) signals errors such as invalid values for variables, which may cause several conditions in a guarded command to incorrectly hold simultaneously, for instance. The max transition t_3 (similarly for $t_1 1$) will remove all tokens in places p_1, p_2, p_3 and p_5 before it fires. This is equivalent to reset the various variables to correct and consistent values and clean up the buffers. For simplity of presentation, the time parameters for this protocol are omitted. Some constraints on the time parameters are obvious. For instance, t_4^* must be at least equal to t_{11}^{**}.

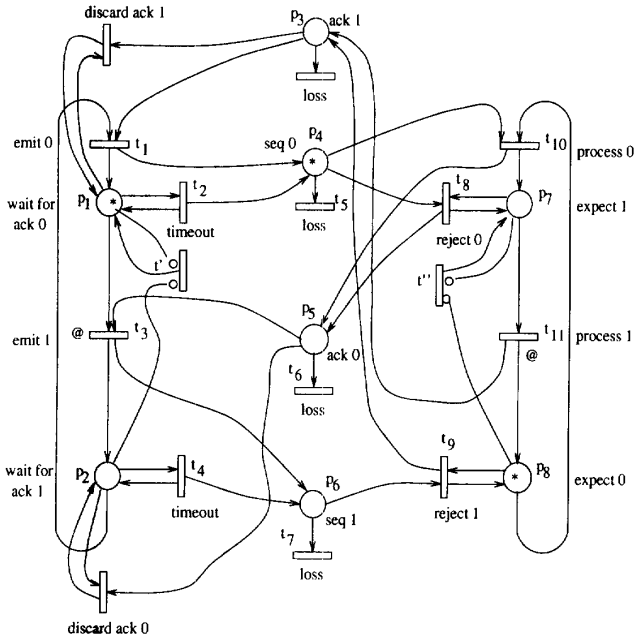

Figure 5: A self-stabilizing extension of sliding window protocol

Theorem 5.2 The sliding window protocol in Fig. 5 is self-stabilizing.

Proof: As always, the error of total-loss-of-token is handled by the inhibitor arcs. The error of creation of arbitrary tokens is solved through the max transitions t_1 and t_3. If there is more than one token in places $p_1, ...$, the firing of transition t_1 will remove all tokens in the places. □

It is interesting to point out that Petri nets with max transitions are strictly less powerful than Petri nets with inhibitor arcs. This is evidenced by the fact that the language L^*, where $L = \{a^n b^n | n > 0\}$, cannot be generated by any Petri nets with max transitions. In fact the class of ordinary Petri nets added with max transitions is not closed under the Kleene closure operation. Details about the modeling power of max transitions are being given in a forthcoming paper in [12].

6 External and internal interferences

As evidenced from the previous sections, indiscriminated use of inhibitor arcs and/or max transitions may cause extreme difficulty in implementing self-stabilizing protocols. The concept of "decentralized control" used in Section 4 is intended to remedy the problem. We formally propose here the concepts of *external* and *internal* interferences, which will allow us to formally quantify the problem.

In general, a component (e.g. a sender, receiver, or a channel) within a communication protocol may be represented by several places and transitions in a Petri net. Let C_1 and C_2 be two communication components that are not physically linked. Informally, an external interference occurs when one place (transition, resp.) belonging to C_1 is forced to connect to a transition (place, resp.) belonging to C_2 by some arc added during the process of self-stabilizing the net. This point has been stated in Section 4, where we argued that the self-stabilizing extension shown in Fig. 2 is very difficult and expensive to implement. However, controlled use of inhibitor arcs, i.e. the arcs that don't cause external interference, such the net shown in Fig. 3, may minimize the implementation difficulty. Similar arguments hold for the use of max transitions. For instance, although the use of max transitions in Fig. 4 gives a more concise representation, implementing the max transition t_1 in that figure is quite difficult and expensive. On the other hand, the two max transitions in Fig. 5 are quite easy to implement because they involve only places local to the sender or receive process.

A place or transition in a net is said to be *internal* if it is only connected to transitions or places belonging to the same component. An internal interference occurs when the internal states of one or more components are forced to be connected to the outside during the self-stabilization extension procedure. The first protocol given in [2] is an example of internal interference, where the state change of a process depends on the internal states of one of its two neighbor processes. On the other hand, the sliding window protocol in Fig. 4 does not suffer from neither external nor internal interference, because the max transition t_3 (t_{11}, resp.) does not refer to any places or transitions in the receiver (sender, resp.) at all. The communication channels there are modeled by four places and none of them is internal.

Both external and internal transitions will cause difficulty in practical implementation, thus should be avoided.

7 Discussions

Using Petri nets as the concurrent model, we discussed the issue of self-stabilizing communication protocols. We argued that self-stabilizing extensions of

communication protocols should not interfere with their normal operations. We also argued that communication protocols are usually delay sensitive and *time-Petri nets* is a more suitable formal model (than ordinary Petri nets) to express self-stabilizing property. The possibility and suitability of using Petri nets with *max transitions* to express self-stabilizing communication protocols were explored.

As pointed by Ghosh in [7], the use of inhibitor arcs to represent self-stabilizing Petri nets seems unavoidable. This may raise the question about the tractability and suitability of Petri nets for self-stabilizing communication protocols. We feel that indiscriminated use of inhibitor arcs will at least cause extreme difficulty in implementing self-stabilizing protocols. This point has been stated in Section 4, where we argued that the self-stabilizing extension shown in Fig. 2 is very difficult and expensive to implement. However, controlled use of inhibitor arcs, such the net shown in Fig. 3, may minimize the implementation difficulty. Similar arguments hold for the use of max transitions. For instance, although the use of max transitions in Fig. 4 gives a compact representation, implementing the max transition t_1 in that figure is quite difficult and expensive. On the other hand, the two max transitions in Fig. 5 are quite easy to implement because they involve only places local to the sender or receive process. The issue of using generalized time-Petri nets to specify and verify communication protocols is worth of further investigation.

It should be pointed out that the notion of *recoverability* and *recoverable communication protocols* proposed by Merlin and Farber in [14] is quite close to (although weaker than) the self-stabilization concept.

Both the nets in Fig. 2 and Fig. 3 are asymmetric. This is in agreement with the conclusions in [5]. In particular, it is not difficult to see that there is no symmetric version of self-stabilizing extension of the net shown in Fig. 3 (trying to come up with such an extension will easily reveal the difficulty and impossibility).

The fact that the self-stabilizing extension in Fig. 3 can only operate at half of the speed of the original net indicates that self-stabilization is usually expensive to implement. This also implies that self-stabilization may be a property too strong and too expensive for certain applications. The bound on the convergence rate for the self-stabilizing extension in Fig. 3 is new and it is not known if this is the optimal bound for this type of delay sensitive rings.

The issue of self-stabilizing *colored Petri nets* is another interesting and important issue. Many complex concurrent phenomena are very difficult, if not impossible, to be modeled by low-level Petri nets. Some others cannot be adequately described by low-level Petri nets at all. Research in the area of *high-level* Petri nets is intended to remedy this problem [11]. High-level Petri nets are in general more expressive than its low-level counter parts and thus often give more succinct and manageable specifications. More importantly, all the formal analysis techniques developed for low-level nets have been easily extended to high-level nets.

The class of *Colored Petri nets* (CPNs) is one of the most influential types of high-level nets. Informally, in a CPN, tokens are differentiable – they are typed (i.e. *colored*). Each place now is associated with a *color set* – the set of types of tokens that can legally be in it. Transitions are now guarded with boolean expressions. A transition can fire provided that (1) it is enabled in the usual sense (for low-level nets) and (2) the associated guard expression is evaluated to true. Each arc is associated with an *arc expression* which regulates conditions under which certain types of tokens can be removed (from input places) or produced (to output places). Detailed definition can be found in [10].

Fig. 6 (a) is a CPN that models the one-bit sliding window protocol. The color set associated with each place used is obvious and the explanation is omitted. Compared with Fig. 5, the CPN is much smaller and succinct. A self-stabilizing extension of the CPN P is shown in Fig. 6 (b), where the input set for the max transitions t_1 and t_6 are $\{p_1, p_3\}$ and $\{p_2, p_4\}$ respectively.

Further research is needed tp extend existing methodologies for low-level Petri nets to CPNs and develop new methodologies suitable for CPNs.

References

[1] F.Bastani, I.Yen, and I.Chen, *Class of inherently fault-tolerant distributed programs.* IEEE Trans. on Software Engi., Vol. 14(10), Oct. 1989, pp.1432-1444.

[2] G.M.Brown, M.G.Gouda, and C.Wu, *Token systems that self-stabilize.* IEEE Trans. on Software Engi., Vol. 14(6), June. 1989, pp.845-852.

[3] J.Burns and J.Pachl, *Uniform self-stabilizing rings.* ACM Trans. on Programming Lang. and Systems, Vol.11, No.2, 1989, pp. 330-344.

[4] L.Cherkasova, R.Howell, and L.Rosier, *Bounded self-stabilizing Petri nets.* Proc. of the 12th Annual Petri Net Conference, Dec. 1991.

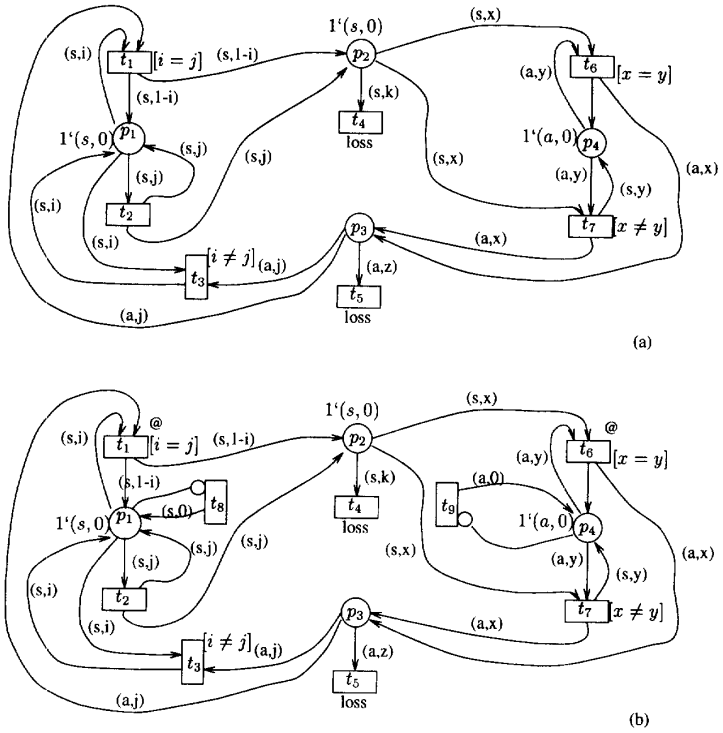

Figure 6: A CPN modeling the one-bit sliding window protocol: (a) the one-bit sliding window protocol; (b) the corresponding self-stabilizing extension

[5] E.W.Dijkstra, *Self-stabilizing systems in spite of distributed control.* Comm. of ACM, Vol. 17(11), Nov. 1974, pp.643-644.

[6] M.Ghodsi and K.Kant, *Well-formed generalized task graphs.* Proc. of 1991 IEEE International Conf. on Parallel and Distributed Systems, Dec. 1991.

[7] S.Ghosh, *Stabilizing Petri nets.* Proc. of 1991 IEEE International Conf. on Parallel and Distributed Systems, Dec. 1991.

[8] M.G.Gouda and N.J.Multari, *Stabilizing communication protocols.* Technical Report, TR-90-20, Dept. of Comp. Sci, The Univ. of Texas at Austin, June 1990.

[9] M.G.Gouda, R.Howell and L.Rosier, *The instability of self-stabilization.* Acta Informatica, Vol 27, Sept. 1990, pp. 697-724.

[10] K.Jensen, *Coloured Petri nets: a high-level language for system design and analysis.* In Advances in Petri nets 1990. LNCS 483, Springer, 1991, pp.342-416.

[11] K.Jensen and G.Rozenberg (eds.), *High-level petri nets: theory and practice.* Springer, 1991.

[12] K.Kant and W.Peng, *On the modeling power of max transitions and high-level Petri nets with max transitions.* Under preparation.

[13] H.S.M.Kruijer, *Self-stabilization (in spite of distributed control) in tree-structured systems.* Information Processing Letter, Feb. 1979, pp. 91-95.

[14] P.M.Merlin and D.J.Farber, *Recoverability of communication protocols - implecations of a theoretical study.* IEEE Trans. on Comm., Vol.COM-24, Sept. 1976, pp. 1036-1043.

[15] J.Peterson, *Petri nets theory and the modeling of systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[16] A.S.Tanenbaum, *Computer networks.* Prentice-Hall, Englewood Cliffs, NJ, 1989.