8 More about GSPN Models

In this note we will consider two simple systems modelled by GSPN and in the course of doing so examine more closely the dynamics of these models with respect to timed and immediate transitions. At the end of the note we will summarise some of the key features of the `PIPE` modelling tool and describe how it may be used to generate and solve GSPN models.

8.1 System Dynamics

Recall that in an untimed Petri net the firing of a transition corresponded to an event in the system. In GSPN we recognised that these events might arise in two different ways:

- they may be induced by the completion of some activity; these activities can be further divided into two categories:
  - those which take significant time to complete.
  - those whose duration is negligible compared with other actions in the system, often called *control actions*.
- they may result from the verification of some logical condition of the system.

Timed transitions represent actions which take some time to be completed. Immediate transitions represent logical events and control actions.

For timed transitions, when they become enabled, i.e. all the input places of the transition become marked with the necessary number of tokens and any inhibitor places are unmarked, we imagine the corresponding activity starting in the system. This activity will continue until it is completed or interrupted. We assume that each transition has a timer or local clock associated with it: when the transition is enabled the timer is set to a value. Since we are using a random variable to represent activity durations, this value is *sampled*[1] from the exponential distribution with the appropriate rate parameter. While the transition is enabled we assume that the timer decrements at a constant speed. When the timer reaches zero the transition fires (assuming that it has remained enabled throughout the clock period).

For immediate transitions, the firing of the transition takes no time and is assumed to occur instantaneously. We also assume that enabled immediate transitions always have priority over any enabled timed transitions, so we fire all possible immediate transitions first. If only one immediate transition is enabled, it fires, and the new marking is produced. If several immediate transitions are enabled, some mechanism is necessary to identify which transition will fire. In fact the choice of transition to be fired is only important in cases where there is conflict between the enabled transitions, i.e. the firing of one will disable another. If the enabled immediate transitions are concurrent, they can be

[1]*Sampling* means randomly choosing a value, in such a way that if you kept on doing this over time and kept track of the values they would satisfy the probability distribution function. Sampling will be treated in more detail when we study simulation. Note that for GSPN models solved as a Markov process this is just a way of thinking about the dynamics of the system, we do not actually sample the distributions.

fired in any order. When enabled immediate transitions are in conflict GSPNs associate *weights*, or relative probabilities, with each transition. We can think of a distribution being sampled to determine which transition actually fires in each case.

Bearing this view of the system dynamics in mind we will consider again the two-processor shared memory system from lecture note 5. For convenience it is shown again in Figure 13.

8.1.1 Example: The multiprocessor system revisited

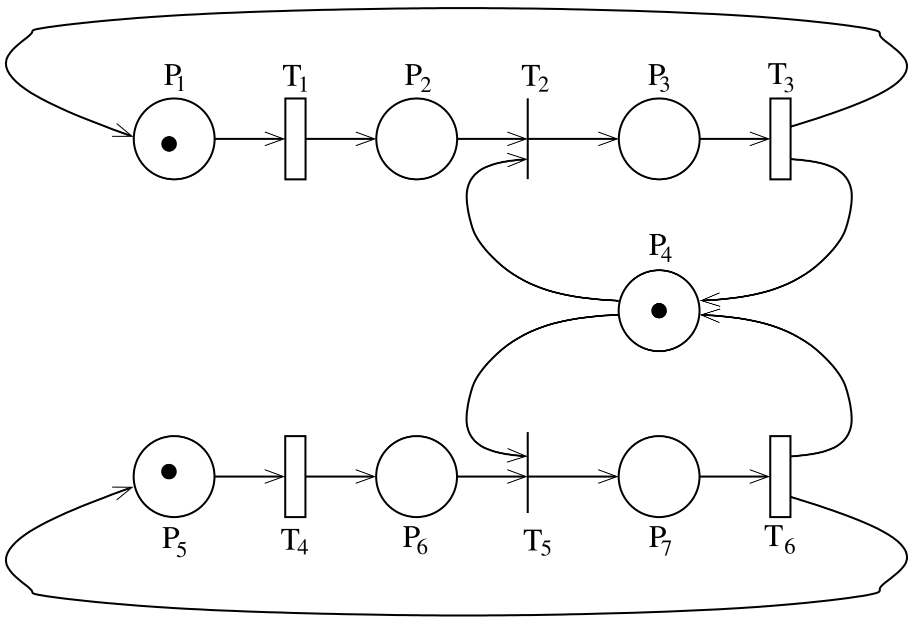


Figure 13: GSPN representing the two-processor shared memory system

In the initial marking of the system we see that both transitions T_1 and T_4 are enabled. Thus we think of them sampling their respective exponential distributions, with parameters λ_A and λ_B, and obtaining values V_A and V_B say. Which transition fires first will be determined by whose associated timer reaches the value zero first—this is called the *race policy.* If $V_A < V_B$, transition T_1 will be the one to fire and the marking changes as $(1, 0, 0, 1, 1, 0, 0) \to (0, 1, 0, 1, 1, 0, 0)$. If $V_B < V_A$, transition T_4 fires and the new marking is $(1, 0, 0, 1, 0, 1, 0)$. The case of $V_A = V_B$ can only occur with probability zero[2] so we do not need to worry about it. Of course each time the distributions are sampled different values may be obtained, and so which transition fires first will vary.

In either case we need to consider what happens to the remaining time on the timer associated with the transition which did not fire. There seem to be two possibilities: either it is forgotten or it is remembered. For example, in the two-processor system if $V_A < V_B$ and transition T_1 fires, the time remaining on the timer associated with T_4 will

[2]The probability that a sample extracted from an exponential distribution takes any specific value v is always zero, so if we know that the value sampled by the first timer is V_A the probability that the second timer will sample the same value is zero.

be $V_B - V_A$. Since T_4 remains enabled after T_1 has fired it would seem unreasonable for time the system has already spent on the corresponding activity to be forgotten. In contrast, if T_1 and T_4 had been in conflict, so that the firing of T_1 removed the enabling tokens for T_4 it might seem reasonable to forget the time that has already been spent on the corresponding activity, and imagine the system starting that activity again from the beginning whenever T_4 next becomes enabled. Note that in the case of conflict between timed transitions which transition will fire is always resolved by the race policy.

Fortunately, the memoryless characteristics of the exponential distribution mean that we do not need to differentiate between the situations when a timed transition should forget its previous work and those when it should remember. Indeed, whether the timer is reset or not, the probability distribution of the time remaining until the firing of a transition will always be the same, and distributed according to the exponential distribution when the transition was first enabled. Thus we can model either of these situations in the same way in a GSPN.

Despite initial impressions from inspecting the net, the two immediate transitions in the GSPN of the two-processor system can never both be enabled at the same time. Thus at most one immediate transition is enabled in any marking and this transition will fire instantaneously leading to the following marking. For example, if T_1 fires as described above, leading to the marking $(0,1,0,1,1,0,0)$, transition T_2 is now enabled and fires at once, leading to the marking $(0,0,1,0,1,0,0)$.

8.1.2 Example: The reader-writer system

The GSPN in Figure 14 is a representation of what is sometimes referred to as the *reader-writer system*. The model represents a system in which there is a set of processes who share access to a common database. On any particular access a process may wish to perform a read or a write. Any number of readers may access the database concurrently; in contrast, a writer requires exclusive access to the database. In between accesses to the database each process will undertake some processing independently.

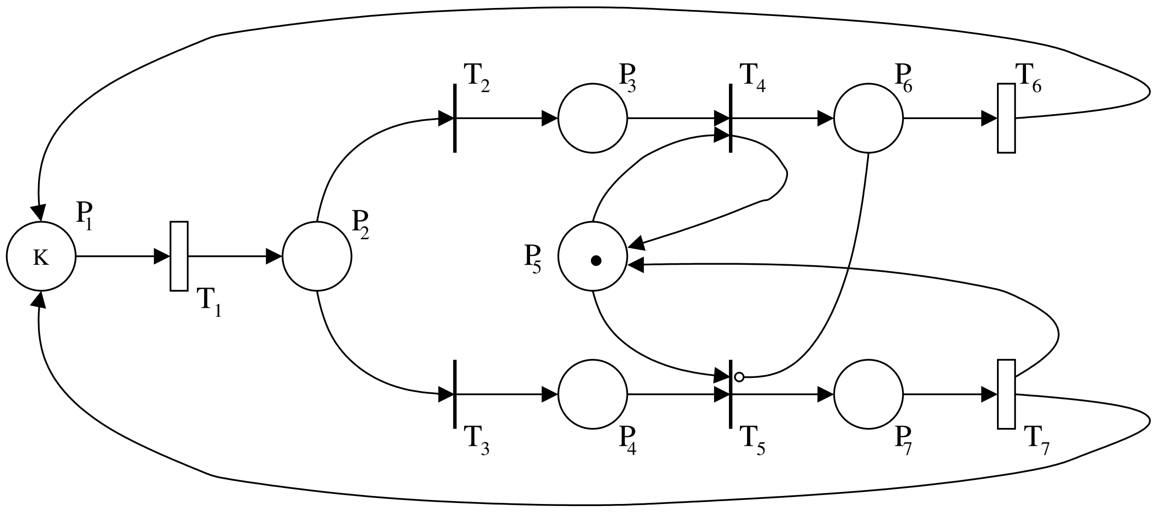


Figure 14: GSPN representing the simple reader-writer system

Timed transitions are used to represent the reading and writing actions as well as the local activities performed by the processes between two subsequent accesses to the database. Immediate transitions are used to implement the decision of which action a process wants to perform on the database (T_2 and T_3) and to implement the scheduling between reading and writing activities (T_4 and T_5). The interpretations of the places and transitions in the model shown in Figure 14 are given below.

- P_1 represents a process undertaking local processing.
- P_2 represents a process ready to commence access to the database.
- P_3 and P_4 model processes ready to read and ready to write respectively.
- P_5 enforces unique access to the database for writing; it can be thought of as representing the condition *no write access is in progress.*
- P_6 and P_7 represent processes currently reading or currently writing respectively.
- T_1 models the independent processing, not related to the database access; each process undertakes local processing for a mean duration of $1/\lambda$ milliseconds.
- T_2 and T_3 model the "decision" of the process to make a read access or a write access; the switching probabilities are chosen to reflect the relative frequencies of these types of access.
- T_4 represents a process gaining access to the database to start a read access.
- T_5 represents a process gaining access to the database to start a write access; note that the transition is inhibited whenever a read is already in progress (P_6).
- T_6 models the database read activity; the average read access lasts for $1/r$ milliseconds.
- T_7 models the database write activity; the average write access lasts for $1/w$ milliseconds.

This system is commonly used in texts about modelling with Petri nets because it is small and easily understood but yet it exemplifies three interesting features which are often important in models: *concurrency of events* (two or more processes may be concurrently accessing the database for reading), *mutual exclusion* (only one process at a time may access the database for writing) and *choice* (an access can either be a read or a write). This version of the model is based on the one presented in the book by Ajmone Marsan *et al.*[3]; in other books variations on this model appear.

Note that the initial marking of the GSPN has a single token in place P_5, representing that the condition is currently satisfied, and a parameter K in place P_1. The *parametric marking* represents the situation that there is some number of processes, K, within the system, but the value of K has not yet been decided. A value must be assigned to K before the model can be solved because, as we have already seen, the number of tokens in the initial marking can drastically change the reachability set of a model.

Let us first consider the immediate transitions of the model. Whenever there is a token in place P_2 the two immediate transitions T_2 and T_3 are both enabled. These transitions

[3]*Modelling with Generalized Stochastic Petri Nets*, M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis, John Wiley, Series in Parallel Computing, 1995

are in conflict because the firing of one will disable the other. Thus weights, or relative probabilities, must be assigned to the transitions to resolve the conflict. If we know that 60% of database accesses are read accesses and 40% are write accesses, we associate weights 0.6 and 0.4 with T_2 and T_4 respectively.

If transition T_2 fires, and P_5 contains a token, transition T_4 is now enabled and fires immediately—no other immediate transitions are enabled and the immediate transition takes priority over any timed transitions which might be enabled, such as T_1. Thus two immediate transitions may fire one after another without any time progressing in our model. If T_3 fires a token appears in place P_4, but transition T_5 will only be enabled if there is no token in place P_6—this is the effect of the inhibitor arc—and there is a token in place P_5.

Let us now consider the timed transitions of the model. If there are K tokens in place P_1 it means that there are K processes currently involved in local processing, each progressing at a rate λ. Thus the total rate at which processes complete local processing is $K \times \lambda$. In the model we represent this by saying that the rate of the transition T_1 is dependent on the marking of transition P_1: the rate of the transition is λ multiplied by the number of tokens in P_1 (by the superposition principle). Such transitions are said to have *marking dependent firing rate.*

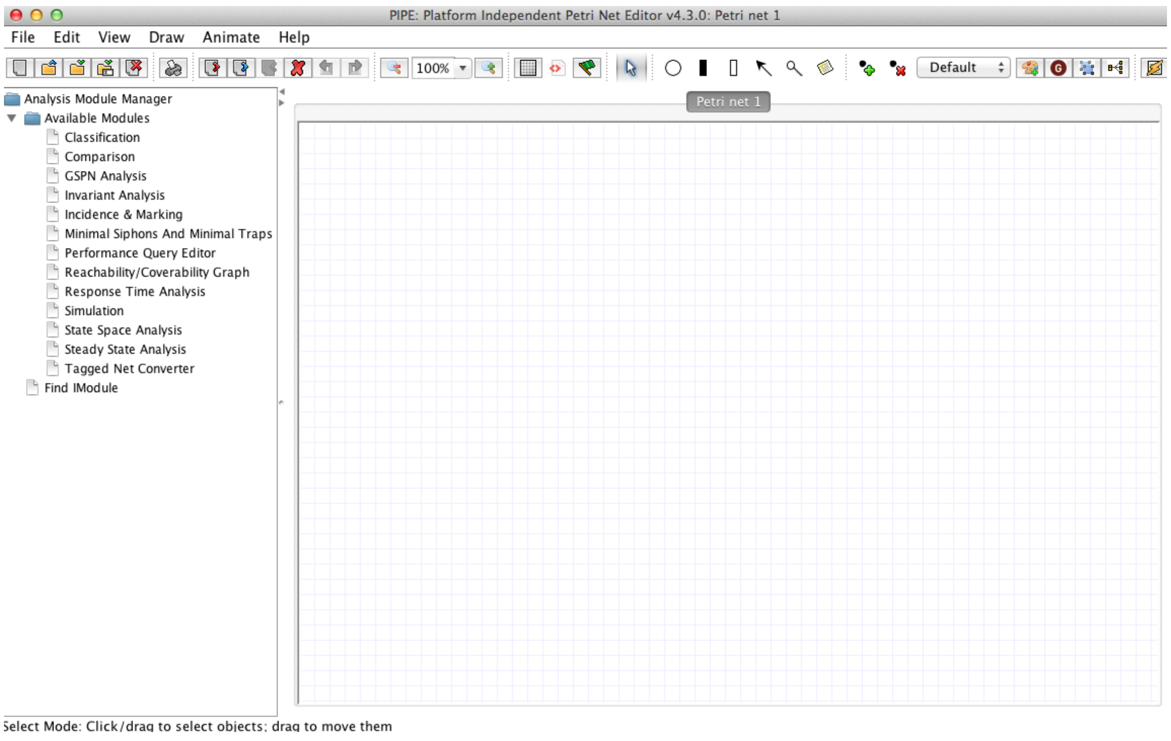
Similarly when more than one process is engaged in a read access, the rate at which a single read access is completed will depend on the number of processes currently reading—the marking of place P_6—and the rate at which reading occurs—the transition rate of transitions T_6 (r).

In general, whenever a transition's enabling condition is satisfied more than once, e.g. K tokens in place P_1 when only one is required to enable transition T_1, there are two possibilities with respect to the firing rate of the transition. We can imagine the transition progressing each set of enabling input tokens *serially.* In this case we only consider the transition to be enabled or not, and when it is enabled it always works at the same rate. This is called *single server semantics* and it is the default case for GSPN models. Alternatively we can think of the transition progressing each complete set of enabling input tokens *concurrently.* In this case the rate at which the transition works will depend upon its *enabling degree.* This is called *infinite server semantics* and in GSPN models it is achieved by setting a marking dependent firing rate.

In the reader-writer model the enabling degree of the transition T_1 in the initial marking is K. Similarly the enabling degree of transitions T_6 can have any value between 1 and K. In contrast, enabling degree of T_5 is at most 1: even though the place P_4 may have up to K tokens, the enabling degree will not be greater than 1 because there is only one token in place P_5.

8.2 The PIPE Modelling Tool

PIPE (Platform Independent Petri net Editor) is an open source, platform independent tool used of the creation and analysis of Petri Nets, and some of their extension, developed at Imperial College. It is implemented in Java and has a graphical user interface, which makes it very straightforward to use. The most recent version is PIPEv4.3.0 and it is recommended that this is the version you install, as earlier versions did have some


PIPE: Platform Independent Petri Net Editor v4.3.0: Petri net 1

Select Mode: Click/drag to select objects; drag to move them

Figure 15: Image of PIPE tool on opening

problems with the delete function and consistency of the graphical user interface and the internal representation. PIPEv.4.3.0 is available for download from
`http://sourceforge.net/projects/pipe2/files/PIPEv4/PIPEv4.3.0/`

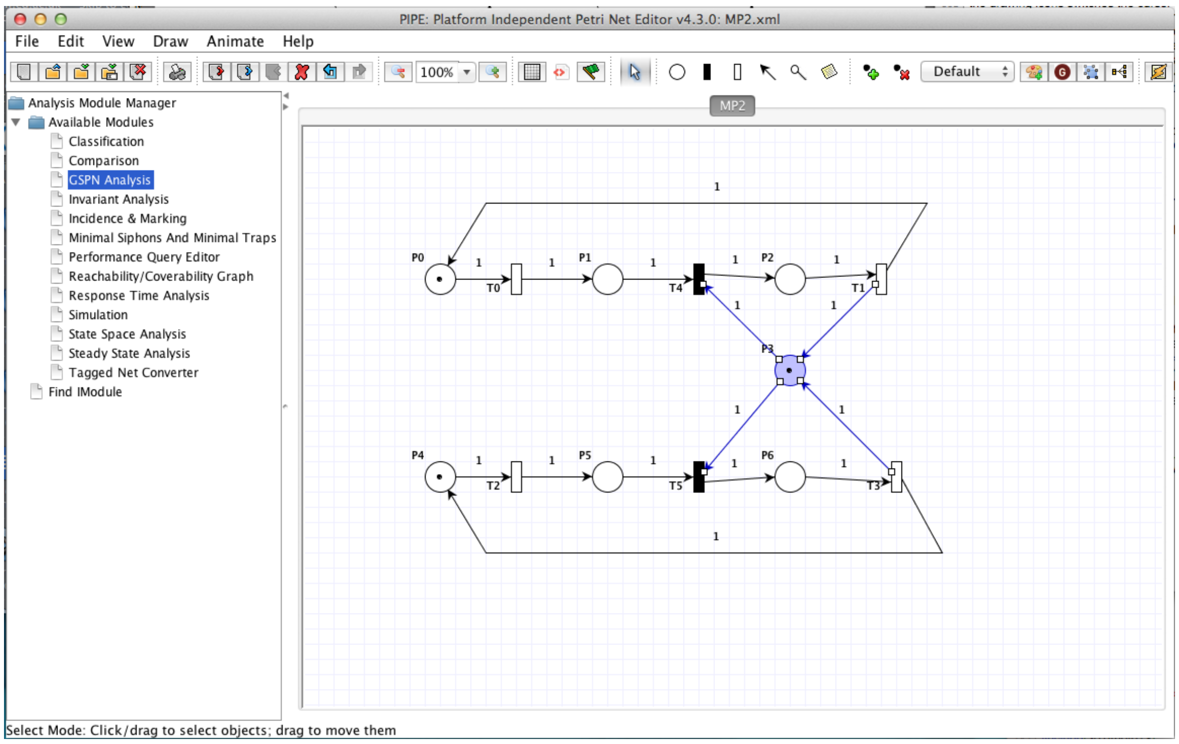
Once you have unpacked the directory/folder `PIPEv4.3.0`, enter that directory and issue the command

./launch.sh or .\launch.bat

according to your operating system, to launch the PIPE tool. This will open window like the one shown in Figure 15.

The large pane on the right is a drawing canvas, and by default the tool is in drawing mode. The elements of a Petri net (places, immediate transitions, timed transitions, arc etc) are all displayed as icons above the drawing canvas. Immediate transitions are represented by dark filled boxes and timed transitions by white boxes. To place an element on the canvas simply click on the icon and then click on the canvas. Places and transitions must be in place before arcs can be added connecting them. Arcs can start and finish on either side of transitions, and bends can be added to an arc by clicking on a position on the canvas that does not correspond to any element.

Once the basic net structure is completed, tokens can be added to indicate the initial marking of the GSPN using the icons to the right of the drawing icons. The dot with a green plus can be used to add tokens while the dot with the red cross icon can be used to delete tokens. If you want to change the layout of the GSPN as displayed on the canvas, clicking on the arrow icon to the left of the drawing icons switches the cursor to selection mode. Now clicking on an element will allow the user to move it. Double clicking on



Select Mode: Click/drag to select objects; drag to move them

Figure 16: Image of PIPE tool modelling the multiprocessor example as a GSPN

a transition will open a popup menu that allows the characteristics of the transition to be set. For example, for a timed transition the rate and the server semantics may be specified. It is also possible to change the name of the transition, or the orientation of its display. Double clicking on a place allows its name to be changed and provides an alternative way to set the number of tokens in the initial marking. Double clicking on arcs allows their multiplicity to be set. Elements may be deleted via the delete icon which is the large red cross icon immediately below the Help menu.

Figure 16, shows PIPE being used to represent the GSPN representation of the multiprocessor example with two processors. Models can be saved, as an `.xml` file, via the `File` menu.

PIPE offers a variety of analysis tools which are listed as modules on the left hand side of the drawing canvas. The one that is most relevant to us is GSPN Analysis. Double clicking on this will pop up a separate window that asks for confirmation of which GSPN is to be analysed; by default it will be the model currently displayed in the canvas but it is also possible to specifyy another model from a file. Once this is confirmed analysis will be carried out, and results displayed in the same popup window. The GSPN analysis displays

- the list of tangible markings,
- the steady state probability distribution over those markings,
- the average number of tokens per place,

- the probability density over each place showing the likelihood of each possible number of tokens in that place,
- the throughput of each timed transition in steady state,
- the average sojourn time in each tangible marking in steady state.

8.3 A GSPN model of the PC LAN with 4 nodes

The model below is a GSPN representation of the PC LAN with four nodes considered in lecture note 4. Note that the inhibitor arc from place P_{i2} to transition T_{bi} (for $i = 1, \ldots, 4$) ensures that a token does not bypass a PC in which a data packet is waiting for transmission. The source file can be found on the course web page as `PCLAN4.xml`.

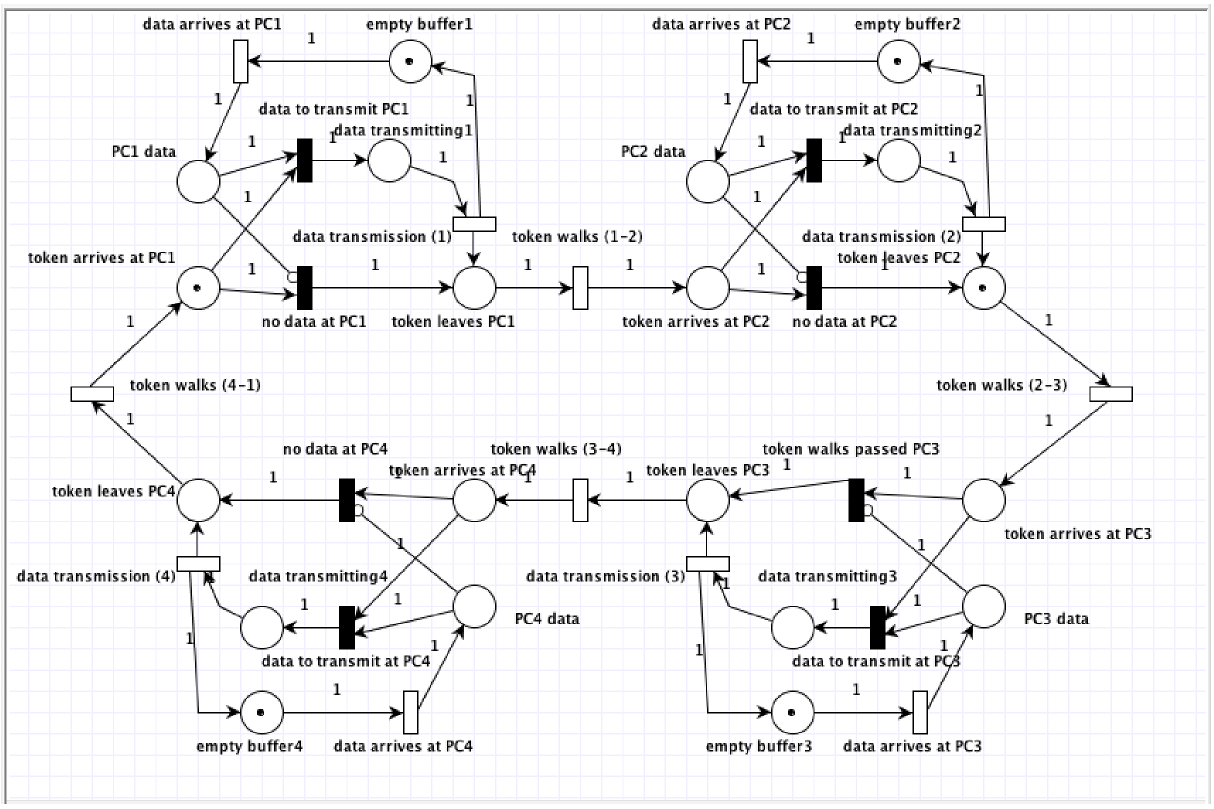


Figure 17: GSPN representing the PC LAN with four nodes

Jane Hillston ⟨Jane.Hillston@ed.ac.uk⟩. February 2, 2017.