4 More Complex Markov Processes

4.1 Introduction

In this note we consider another example Markov process, modelling a more realistic system than those considered in the previous lecture notes. However we will quickly see that even for such moderately sized examples difficulties soon arise when we work directly at the level of the Markov process. Several attempts may be necessary before a correct characterisation of the state space is achieved; generating the state space and calculating the transition rates is time consuming and error prone; interpreting the results and deriving the performance measures is also a complex task. The last two problems are helped by using a high-level modelling paradigm, as we will discuss in the last section of the note.

4.2 Upgrading a PC LAN

4.2.1 Description of the system

The problem we consider is to determine the mean waiting time for data packets at a PC connected to a local area network, operating as a token ring. Such a network uses a transmission medium that supports no more than one transmission at any given time. To resolve conflicts, a token is passed round the network from one node to another in round robin order. A node has control of the medium, i.e. it can transmit, only whilst it holds the token. In a PC LAN every PC corresponds to a node on the network. Other nodes on the network might be peripheral devices such as printers or faxes but for the purposes of this study we make no distinction and assume that all nodes are PCs. There are currently four PCs (or similar devices) connected to the LAN in a small office, but the company has recently recruited two new employees, each of whom will have a PC. Our task is to find out how the delay experienced by data packets at each PC will be affected if another two PCs are added.

Each PC can only store one data packet waiting for transmission at a time, so at each visit of the token there is either one packet waiting or no packet waiting. The average rate at which each PC generates data packets for transmission is known to be λ (see Figure 2). We also know the mean duration, d, of a data packet transmission, and the mean time, m, taken for the token to pass from one PC to the next. It is assumed that if another data packet is generated, whilst the PC is transmitting, this second data packet must wait for the next visit of the token before it can be transmitted. In other words, each PC can transmit at most one data packet per visit of the token.

4.2.2 Modelling the system

The first stage of developing a Markov process to represent the system is to choose the random variables which will be used to characterise the **states** of the system. Firstly, considering the nodes, we can think of each PC as being in one of two states:

- the PC does not currently have a data packet to transmit; or

- the PC has a data packet waiting at the interface for transmission.

We map these two observations to the values 0 and 1 respectively.

For the token we can think of its current state being characterised by its current position. Thus, if there are N PCs in the network the states of the token correspond to the values $\{1, 2, \dots N\}$.

It follows that when there are N PCs in the network each state can be represented as a $(N+1)$-tuple—one element for the state of each PC and one for the position of the token.

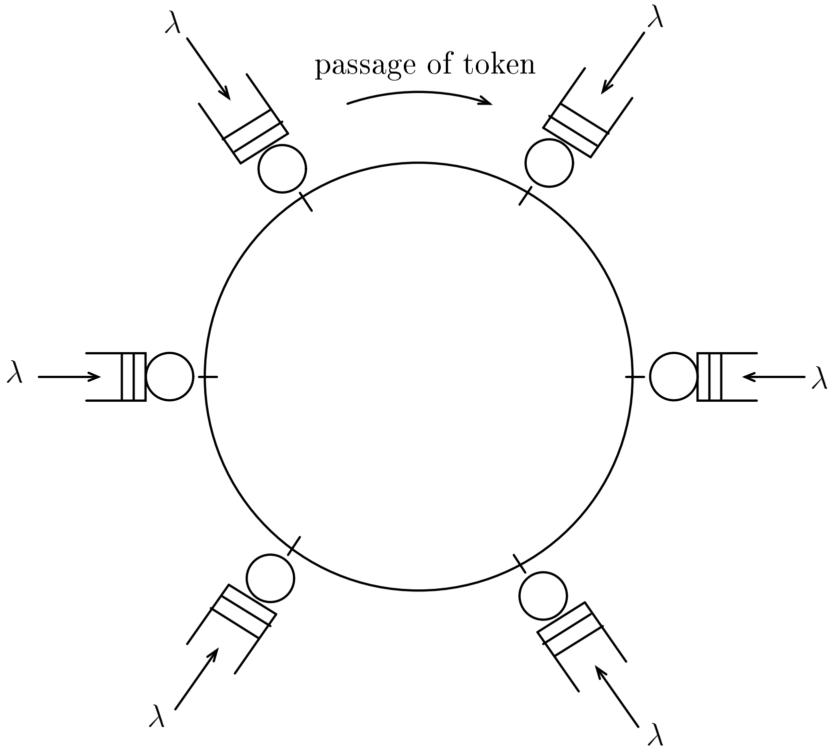

Figure 2: A six-node token ring network

The second stage is to decide on the **events** we wish to represent, and to assign parameters to the corresponding exponential distributions based on the information we have been given about the system. In this case, the events will be:

- the arrival of data packets—we know that at each PC data packets are generated at a rate of λ;
- the movement of the token between PCs—we set the rate of the movement, ω, to be the inverse of the mean time taken $\omega = 1/m$;
- the transmission of data packets—similarly, we set the rate of transmission, μ, to be the inverse of the mean transmission time $\mu = 1/d$.

It is always a good idea to test our reasoning at an early stage. In this case the most sensible approach is to consider a very simple version of the problem, for example, a network with just two PCs. For this system the states would be represented by the triple:

$$(\underbrace{\mathrm{PC}_1}_{\{0,1\}}, \underbrace{\mathrm{PC}_2}_{\{0,1\}}, \underbrace{\mathrm{token}}_{\{1,2\}})$$

where each of the elements has the possible values shown. This would give a model with eight states:

$$\begin{array}{cccc} (0,0,1) & (0,1,1) & (1,0,1) & (1,1,1) \\ (0,0,2) & (0,1,2) & (1,0,2) & (1,1,2) \end{array}$$

Let us consider the state $(1,1,1)$: the token is at PC_1 which has a data packet to transmit. The description of the system in Section 4.2.1 implies that in this state the only possible event is the transmission of PC_1's data packet. The state becomes $(0,1,1)$. If another data packet is now ready for transmission at PC_1, the state will once again become $(1,1,1)$ and a second transmission will be possible without the token visiting PC_2. This is contrary the behaviour of the system which was described.

The only conclusion is that our current version of the model is unsatisfactory and needs to be modified. There are several possibilities:

- One way to avoid the problem outlined above would be to ensure that after transmission of the data packet from PC_1 the token is moved to PC_2: $(1,1,1) \longrightarrow (0,1,2)$; and similarly after a transmission from PC_2. However, this ignores the waiting time while the token is passed between nodes of the network and amalgamates two events into one.
- Other solutions involve introducing more states into the model so that the state when a transmission has just occurred is distinguished from the state when the token has just arrived at a PC. There are two possibilities for doing this: introducing another state for each PC or introducing N more states for the token.

  In the former case, after transmission the PC would enter a kind of limbo state where it could not accept any new data packets into its interface until the token has moved on. This does not reflect the behaviour of the system as described.

  Alternatively, we can notionally move the token past PC_1 when it has completed a data transmission while recording that it still needs to be passed to PC_2. In this way the token is clearly not available for any more data transmission at PC_1 until it has completed a circuit of the ring, but it still has to be passed to PC_2. This is the solution we adopt.

Thus, the states of the token now become $\{1,\ldots,N,1+,\ldots,N+\}$ where $n+$ denotes the state of having completed a data transmission at PC_n. In the two node network, the states are now represented by triples:

$$(\underbrace{\mathrm{PC}_1}_{\{0,1\}}, \underbrace{\mathrm{PC}_2}_{\{0,1\}}, \underbrace{\mathrm{token}}_{\{1,2,1+,2+\}})$$

and the possible states are:

$$\begin{array}{cccccccc} (0,0,1) & (0,1,1) & (1,0,1) & (1,1,1) & (0,0,1+) & (0,1,1+) & (1,0,1+) & (1,1,1+) \\ (0,0,2) & (0,1,2) & (1,0,2) & (1,1,2) & (0,0,2+) & (0,1,2+) & (1,0,2+) & (1,1,2+) \end{array}$$

Using this new scheme, in general, for N PCs the state of the system will be described by an $(N+1)$-tuple:

$$(\underbrace{\mathrm{PC}_1}_{\{0,1\}},\ \ldots,\ \underbrace{\mathrm{PC}_i}_{\{0,1\}},\ \ldots,\ \underbrace{\mathrm{PC}_N}_{\{0,1\}},\ \underbrace{\text{token}}_{\{1,\ldots,N,1+,\ldots,N+\}}\ )$$

The number of states in the model, when there are N PCs, will be

$$2^N \times (2 \times N) = N \times 2^{N+1}.$$

Of course, in many offices there will be 20 or 30 PCs attached to a LAN rather than just 4 or 6. Even for a straightforward model such as the one we have constructed, systems of this size are beyond the capabilities of available tools to solve analytically. The size of the state space, $|S|$, for various values of N are shown in the table below:

N	2	3	4	5	6	8	10	20	30
$\|S\|$	16	48	128	320	768	4096	20480	4.194304×10^7	6.442450×10^{10}

4.2.3 Solving the model

The models with 4 nodes and 6 nodes were solved using `maple` (see transcript at the end of this note). The transition rates for these models were generated using a Standard ML program and are available from the web pages

`www.inf.ed.ac.uk/teaching/modules/ms/maple`

in `maple` input files `PC-LAN4.trans` and `PC-LAN6.trans` respectively. Similarly, the files `PC-LAN4.table` and `PC-LAN6.table`, provide the assignment of numbers (e.g. $1, 2, \ldots, 128$ and $1, 2, \ldots 768$) to states for each of the models.

The following values were assigned to parameters of the model in both cases (the unit of time is assumed to be milliseconds),

$$\lambda = 0.01 \qquad d = 10 \qquad \mu = 0.1 \qquad m = 1.0 \qquad \omega = 1.0$$

We wish to calculate the average waiting time of data packets at any PC in the network. Since all the PCs are statistically identical, we can arbitrary choose one as representative—we select PC_1. Waiting time, which is the residence time of a data packet at the interface minus the transmission time[1], cannot be derived directly from the steady state distribution since it is neither a state-based nor a rate-based performance measure. However, if we know the average number of data packets at the interface, and the average throughput, we can apply Little's law to calculate residence time and therefore the waiting time.

There will be a data packet waiting in the interface of PC_1 whenever the first entry in the state tuple is "1". Therefore the average number of data packets, $\mathcal{N}$, is the total probability of being in one of these states, i.e. the sum of the values in the elements of the steady state vector corresponding to these states. Similarly, a data transmission from

[1] Note that in the model a data packet remains at the PC until its transmission is complete, there is no state of being in transit. Thus its residence time includes the transmission time.

PC_1 can occur whenever there is a data packet waiting for transmission and the token is at PC_1. Therefore the average throughput X will be the rate at which transmission occurs, μ, multiplied by the total probability of being in one of these states. Finally let $\mathcal{W}$ denote the average waiting time:

$$\mathcal{W} = \frac{\mathcal{N}}{X} - 1/\mu$$

The `maple` expressions for calculating $\mathcal{N}$ and X for each of the models can be found in the files `NumberPC-LAN4`, `ThroughputPC-LAN4`, `NumberPC-LAN6` and `ThroughputPC-LAN6`, which are in the same directory as the files listed above. Based on these expressions and the calculated steady state distributions we find the following values: for 4 PCs

$$\text{average waiting time, } \mathcal{W} = \frac{0.1333}{0.008666} - \frac{1}{0.1} = 15.3862 - 10 = 5.3862$$

for 6 PCs

$$\text{average waiting time, } \mathcal{W} = \frac{0.1719}{0.00828} - \frac{1}{0.1} = 20.7678 - 10 = 10.7678$$

Thus the average waiting time for data packets will almost double when two more PCs are added to the network.

4.2.4 Summary

This example was intended to give you an impression of how even relatively small systems, such as a PC LAN connecting just four machines, can very quickly give rise to models with hundreds of states. You will get a good idea of the complexity of these, still relatively small, models if you look at the Maple files containing all the transitions (see above). Even with just three nodes in the network the model generates 48 states which is large enough to be very difficult to accurately generate by hand. Similarly, trying to extract even straightforward state-based performance measures from models of this size is time-consuming and error-prone.

Exercise: Try to list all the states and draw the state transition diagram for the three node model.

The PC LAN model also illustrates that our first attempt is not necessarily successful. Like any complex problem solving task, modelling is often an iterative process: only when we try to use the model do we become aware of a flaw in the abstraction used. In this case the initial representation of the states of the system was too abstract—we needed more information about the state of the token in order to capture the full behaviour of the system.

4.3 High-level modelling paradigms

Although Markov processes are a convenient modelling technique from some points of view, as noted in the introduction to this note, they can be difficult to work with once the number of states gets into hundreds rather than tens. Since most realistic models of systems involves hundreds of thousands, if not millions of states, constructing the complete state space and finding all the possible transitions becomes a serious problem. Obviously computer-assistance is needed. Moreover, for that assistance to be really worthwhile we need a more abstract description of system behaviour than direct manipulation of the state space.

High-level modelling paradigms are formalisms which provide the modeller with a means of abstracting system behaviour. Now the modeller describes the system in terms of the primitives of the paradigm rather than expressing the model directly in terms of the states and transitions of the Markov process. The formalism includes a procedure for generating the corresponding Markov process from a high-level model.

In this course we will consider three such high-level modelling paradigms, *stochastic Petri nets*, *stochastic process algebras* and *queues* and *queueing networks*. The primitives of stochastic Petri nets and stochastic process algebras are still quite low level, with transitions explicitly represented. In contrast, in queueing networks sophisticated primitives, which encode a lot of information, are used. The benefit of the simplicity of the stochastic Petri net/process algebra approach is much greater expressiveness—these models can capture many situations which cannot be represented using queueing networks.

Performance modelling tools based on high-level modelling paradigms generally provide several important features:

- support for model construction—in some cases this will include a graphical interface, with menus and icons;
- automatic generation of the corresponding Markov process—some tools will even recognise special cases of models which can be efficiently solved during this stage; others will rely on the modeller knowing when special cases apply;
- solution of the Markov process—either using the same basic approach that we have been applying or more sophisticated solutions based on particular model structures;
- expression of performance measures in terms of the high-level model and automatic derivation of the corresponding measures at the Markov process level.

4.4 Maple transcript for model PC-LAN6

```
with(linalg);                                        # load linear algebra package

lambda := 0.01;                                      # set values of parameters

mu := 0.1;

omega := 1.0;

Q := array(sparse,1..768,1..768);                    # declare the generator matrix

b := array(sparse,1..768);                           # declare the solution vector

read `/home/jeh/CS4/MandS/maple/PC-LAN6.trans`; # read in the entries for Q

Qt := transpose(Q):                                  # transpose the generator matrix

for i to 768 do Qt[768,i] := 1.0 od:                 # replace the last equation by
                                                     # the normalisation constant in
b[768] := 1.0;                                       # both Qt and b

p := linsolve(Qt,b);                                 # solve to find the unknowns,
                                                     # the steady state distribution

read `/home/jeh/CS4/MandS/maple/NumberPC-LAN6`;
                                                     # derive the mean number of
                                                     # data packets at PC1

read `/home/jeh/CS4/MandS/maple/ThroughputPC-LAN6`;
                                                     # derive the mean throughput
                                                     # of data packets at PC1

w = N/X - 1/mu;                                      # calculate waiting time as the
                                                     # residence time (N/X) minus the
                                                     # transmission time (1/mu)
```

Please note that you would need to adjust the pathnames to match your own location of the maple input files.