Delft University of Technology

Quantitative Evaluation of Embedded Systems
IN4390

Lab 3(Mandatory Part)

Authors: Jiaxuan Zhang(5258162)
Yiting Li(5281873)
Group ID: (26)

January 21, 2021

1 Question 1

1.1 Node1.py

In `node.py`, each QEES node has: 3 subscribers, one for other nodes' state, one for stop command and one for receiving token; 2 part of publishers, one for publishing own state, another is a set of publishers, each publisher in the set is used to send the token to other nodes; 2 timers, one for generating token and another for removing token.

During the setup state, the program read parameters from the command line/shell script. The shell scripts set 3 nodes: one talker and two listeners. The talker's talking rate is set to 10 tokens/seconds, and listener one and listener two have the same possibility to receive a token. The listener's listening rate is set to 6 tokens/seconds, and each listener maintains queues for each talker. The queue for each talker has five positions. It also sets the talker to send a message to two listeners with $(0.5, 0.5)$ possibility.

When the talker's token generating timer reaches the specified interval, a token will be generated. The talker will send a message to a topic corresponding to a listener according to the given possibility. This message contains the sender ID and one token. When the listener's token removal timer reached the specified interval, it will remove one token from each queue corresponding to different talkers if the queue is not empty.

The subscriber in the listener keeps monitoring the topic to receive the message immediately and put the token into the queue (a listener node maintains a different queue for the different sender). It will then check whether the token in the queue is larger than the maximum queue length. If so, the node will deactivate, and it will notify its state to other nodes.

A deactivated node can do nothing except receiving a token or stopping message. Each talker will set the possibility for a deactivated node to 0, which means no token will be sent to a deactivated node anymore.

1.2 Node2.py

In the node2.py, each QEES node consists of state's and tokens' publisher and subscriber, listener publisher, timer for tokens' publishing and generating, stop command receiver and information about each listener of this QEES node and rate of sending tokens to each listener by this QEES node. The input arguments of each QEES node are listeners information *listener* (including listeners' name and transformation rate), node name *number* (from 1-5), initial queue length *queue_start* (200) and maximum length *queue_max* (250).

During the initialization of the node, the node saves the input parameters and sets the node state to activated. A state publisher and subscriber whose name is *state*, data type is $UInt8$, size is 10 are initialized, the callback function(*state_callback*) for state subscriber will set node's weight to 0(no receiving). Similarly, stop command subscriber, listener publisher, token subscriber and publisher are initialized. A timer for token generating with interval as *time_step* is initialized, the callback function *generation_callback* will send data and record information to log. Similarly, timer for token publishing is initialized.

For sending data: after a certain period of time (*time_step*), when a node is activated and the queue length is greater than the demand of the destination node, the node will send data to the listener. The amount of data sent is determined by the queue length of the current node, a fixed rate $time\_step * 0.01$, and a ratio (predefined variable *rate*). This process is actually simulate the exponential rate of sending.

After a node accepted information from other nodes, it would split the received information, add the received data volume to the original queue. If the current node queue exceeds the maximum limit, this node would become deactivated. When the node is no longer active, it stop sending information, and will inform other nodes of its inactive state, so that no more information will be received by this node.

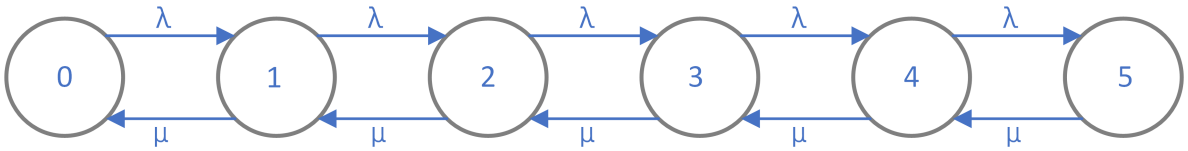

Figure 1: Caption

2 Question 2

2.1 Sub-Question 1

First, we need to make some assumption on the model:

1. Although in `node.py` the token publisher and remover, motivated by the timer, will be activated every fixed time interval and generates/remove one token each time, we still use 2 M/M/1 queues, each with five capacity to model the system. M/M/1 queue means we regard the arrival rate and removal rate as an exponential distribution. We made this assumption because, without exponential distribution, we need to use operational laws, which do not care about distribution, to deal with it. However, one parameter, busy time, is unknown. That makes direct use operational impossible. Moreover, Poisson distribution can be simulated by binomial distribution if n in binomial distribution is very large. The sender process with fixed interval and direction possibility can be regarded as a binomial distribution.

2. For Question 1, we do not care about the deactivated state. If we model M/M/1 queue with Markov Chain and set a deactivated state with an input link without an output link, the chain will be reducible and will not have a steady state.

We model two listener nodes to two M/M/1 queue based on these two assumptions, both with capacity 5. The state model is shown in Figure 1;

When at steady state, the distribution should meet equation 9. And because the receiving rate of each listener is 5 tokens/seconds, and listening rate (removal rate) is 6 token/seconds, $\lambda = 5, \mu = 6, \rho = \frac{\lambda}{\mu}$. Then we can calculate the steady state of the system shown in equation 2.

$$\begin{aligned} \lambda P_{i-1} &= \mu P_i \quad \text{for } i \in \{1,2,3,4,5\} \\ \sum_{i=0}^{5} P_i &= 1 \end{aligned} \tag{1}$$

$$\sum_{i=0}^{5} \dot{P}_0 = 1 \tag{2}$$

$$[P_0, P_1, P_2, P_3, P_4, P_5] = [0.2506, 0.2088, 0.1740, 0.1450, 0.1208, 0.1008]$$

Therefore, we can calculate:

$$\begin{aligned} N = P \cdot Numbers &= \sum_{i=0}^{5} P_i \cdot i = 1.979; \\ X(\text{throughput}) &= \lambda = 5 \\ U(\text{utilization}) &= 1 - P_o = 0.7494 \\ R(\text{response time}) &= \frac{N}{X} = 0.3958 \end{aligned} \tag{3}$$

2.2 Sub-Question 2

The petri net of this system is shown in Figure 2

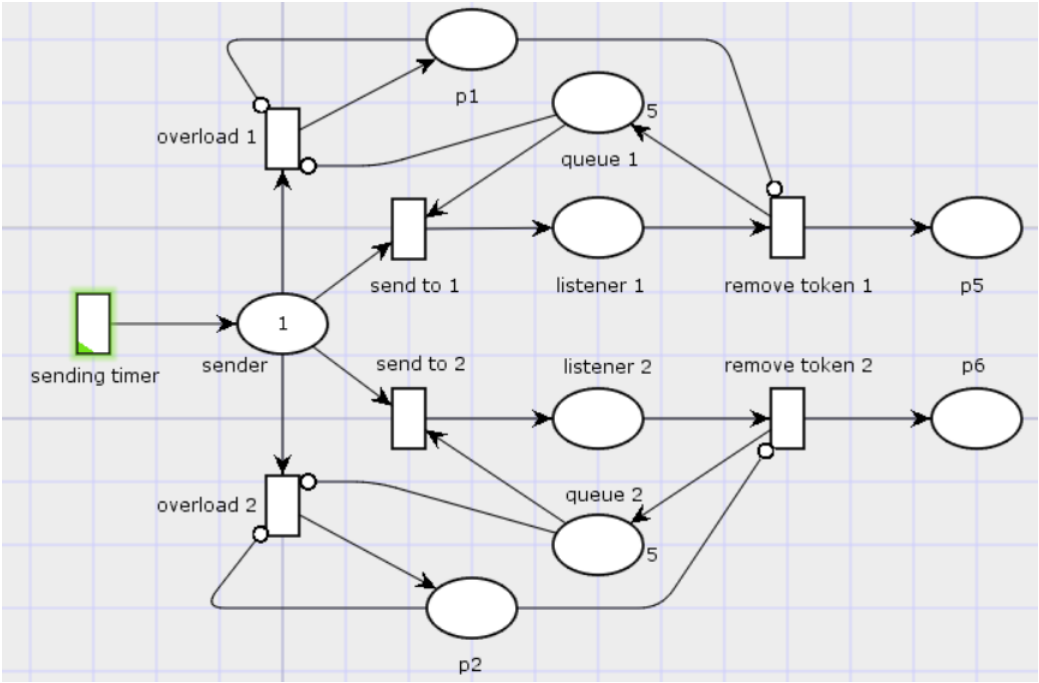

Figure 2: Petri net

In the Petri net, `queue 1` and `queue 2` model the finite queue in each listener, and initially have 5 tokens. Besides, We set an upper bound 1 to place `sender`, that is not set in `node.py`. But in `node.py`, after listeners are all deactivated, the talker will not send any token but will still generate token. So it is rational. `p1` and `p2` are prepared for deactivation. After `queue1` is empty and the sender decides to send the next token to `listener 1`, the only transition is to fire `overload 1` and send a token to `p1`. After `p1` has one token, listener 1 will have no possibility to receive or remove the token. It is the same to `queue2` and `listener 2`.

The system does have a deadlock. The deadlock will happen if `p1`, `p2`, and `sender` all have one token, That means, two listener node has deactivated. According to assumption above, after two listener node has deactivated, the system can not work correctly, although talker can still generate token, we regard it as a deadlock state.

2.3 Sub-Question 3

In Sub-Question 1, we ignored the deactivated state. However, deactivation does exist. When the token in the queue is more than the queue's maximum capacity, the node will deactivate. After a node deactivates, it will stop working, and the talker will only send the token to another listener. If both listeners havehave deactivated, the the token generated in the the talker will no longer be sent to any listener. That means no part of the system can deal with the talker's request. That is not acceptable. Besides, if one listener deactivates, another listener will deactivate soon because the talker's sending rate is 10 token/second, much higher than the listener's removal rate 6 token/second. Therefore, even one listener deactivates is not permitted. That is very important in evaluating the dependability.

Deactivation will happen if one listener has five tokens and receives one more token. Based on the result of Sub-Question 1, the possibility of reaching a state where deactivation will happen immediately

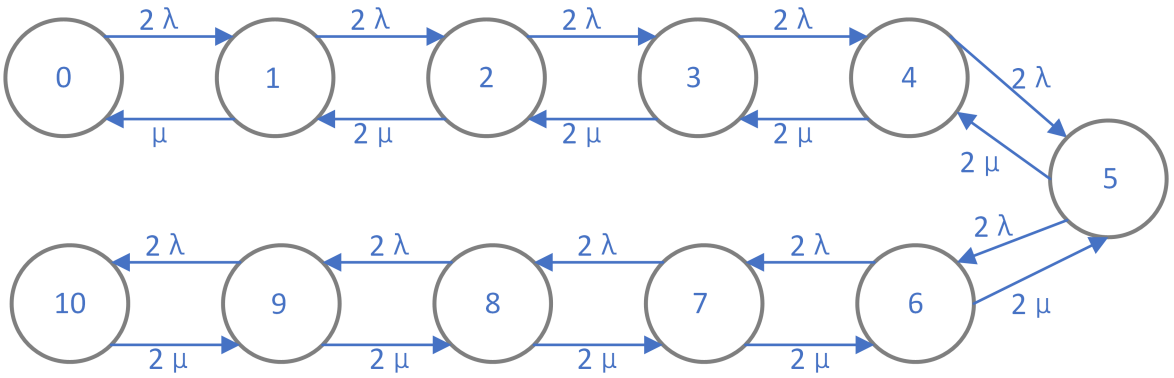

Figure 3: Caption

is shown in equation 4. Therefore the system has 89.2% not arrive an unacceptable result.

$$\begin{aligned} P &= 2 \cdot P(\text{only one of the listener's token} = 5) \cdot P(\text{receive the 6th token}) \\ &\quad + P(\text{two listener's token} = 5) \\ &= 2 \cdot 0.1008 \cdot (1 - 0.1008) \cdot 0.5 + 0.1008 \cdot 0.1008 \\ &= 0.1008 \end{aligned} \tag{4}$$

One way to improve the situation is to combine the queue. After that, the system should be modeled as one M/M/2 model with a capacity of 10. The new jump chain of M/M/2 queue is shown in Figure 3. With M/M/2 model, the steady-state can be modeled as shown in 5

$$\begin{aligned} 2\lambda P_0 &= \mu P_1 \\ 2\lambda P_{i-1} &= 2\mu P_i \quad \text{for } i \in \{2, 3, \cdots 109\} \\ \sum_{i=0}^{10} P_i &= 1 \end{aligned} \tag{5}$$

Therefore, we can calculate the steady distribution ($\lambda = 5, \mu = 6, \rho = \frac{\lambda}{\mu}$) in equation 6. So after modification, the system has 96.56% not arrive an unacceptable result, the dependability is improved.

$$\begin{aligned} (1 + \sum_{i=1}^{1} 02\rho^i) P_0 &= 1 \\ P_0 = 0.1066, P_{10} &= 0.0344 \end{aligned} \tag{6}$$

3 Question 3

3.1 Sub-Question1

The Continues Time Markov Chains(CTMC) model for `node2.py` model is shown in the Figure4. We have 5 nodes who can either receive or sending data with a certain speed. Speed expression is shown in the equation7, where $V_{i,j}$ represent the sending speed from node i to j, and $rate_{i,j}$ is defined in `node2.py`.

$$\begin{aligned} V_{i,j} &= q_i \cdot rate_{i,j} \cdot k; \\ k &= 0.01 time_{step} = 0.005; \\ i, j &\in \{1, 2, 3, 4, 5\} \end{aligned} \tag{7}$$

Before discussing the steady state of the Markov chain, we need to know whether the system is

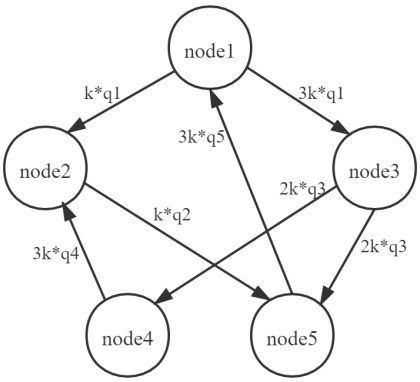

Figure 4: CTMC model for question 3

irreducible, that is, that is, whether the system cannot communicate with other nodes after arriving some certain nodes. In the `node2.py` model, each node has a maximum queue length. When content in the queue exceeds the maximum length, this node would convert to inactivated, and then cannot communicate with other nodes. Intuitively, *node*2 should be the first one the become inactivated, because it seems that the inflow is more than the outflow even q_1, q_2, q_4 might be different. After *node*2 became inactivated, *node*4 has no outflow, and would be filled full soon, then turn to inactivated too. After that, only *node*1, *node*3, *node*5 can communicate with each other, shown in the Figure5.

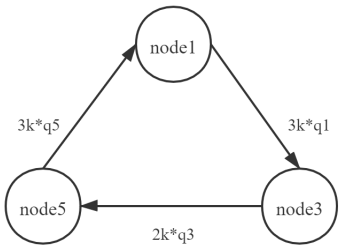

Figure 5: CTMC model for question 3 steady state

The Generator Matrix is 8.

$$G = \begin{bmatrix} -3 & 3 & 0 \\ 0 & -2 & 2 \\ 3 & 0 & -3 \end{bmatrix} \tag{8}$$

Solving the steady states function. Considering that data volume in the system is 1000, 500 of them was stuck in *node*2, *node*4, we can reach the solution for steady states.

$$\begin{gathered} \pi G = 0 \\ \Rightarrow P = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 & P_5 \end{bmatrix} = \begin{bmatrix} 142.9 & 250 & 214.2 & 250 & 142.9 \end{bmatrix} \end{gathered} \tag{9}$$

The final result of `node2.py` is not strictly as in 9, because in coding, we cannot realize it in continuous way.

3.2 Sub-Question2

The reason for normal Petri Nets is partly unusable to `node2.py` model has two aspects. First of all, intuitively speaking, the data sent by nodes is not integer, but defined by an exponential distribution. Theoretically, using discrete tokens can only simulate a continuous variables. If we want to fully express

this model, we have a tremendous computation to complete, which is impossible to finish, because the amount data being transmitting, known as amount of tokens consumed by transition not only depend on the fixed(exponential distribution) sending rate, but also depends on the queue length(from 0 to 250).

Secondly, delay of transitions that are linked to the same place from enabling to firing in normal Petri Nets is unknown, which is opposite to the case in the `node2.py` model. Therefore, using normal Petri Nets is infeasible.

3.3 Sub-Question3

Whether those extensions of basic Petri Nets are able to model the network in the `node2.py` model is discussed below.

Firstly, Colored Petri Nets is unable to model the `node2.py` model. Coloured Petri nets preserve useful properties of normal Petri nets and at the same time extend and introduce color into formalism to allow the distinction between tokens. In the `node2.py` model, tokens have no difference, it stands for a unit of data. Using color to allow the distinction between tokens doesn't help us model a continuous exponential distribution model, therefore, Colored Petri Nets is not fit for this question.

The definition of Stochastic Petri Nets and Timed Petri Nets is given. Timed concept in Petri Nets is associated with the delay from enabling to the firing of a specific transition. Such a Petri net model is known as a timed net if the delays are deterministically given, or as a stochastic net if the delays are probabilistically specified.

Stochastic Petri Nets(SPN) is suitable for modeling the transformation process satisfied exponential distribution that the code want to simulate, but not so suitable for modeling the network setup by code itself. Reasons for that is, first, transformation process satisfies exponential distribution and delays of transitions are probabilistically specified in SPN, so SPN can model this transformation process well. Second, considering that tokens in SPN is integers, but data in network setup by code itself is decimal, so SPN cannot model the actual network built by `node2.py`.

Lastly, Timed Petri Nets is not suitable for modeling. Because, tokens in Timed Petri Nets are integers too, so it is not suitable for actual model. Moreover, the time delays are deterministically given, which is not strictly satisfied exponential distribution. Therefore, the conclusion is that Timed Petri Nets is not suitable for both the transformation process that code wants to simulate and actual network setup by code itself.