Comparison of discrete simulation models' results in evaluating the performances of *M/G/C/C* networks

Noraida A. Ghani[1], Mohd. Kamal Mohd. Nawawi[3], Ruzelan Khalid[3]
Luthful A. Kawsar[1, 4], Anton A. Kamil[1], Adli Mustafa[2]

[1]School of Distance Education, Universiti Sains Malaysia, 11800 Penang, Malaysia
[2]School of Mathematical Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
[3]School of Quantitative Sciences, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah
[4]Department of Statistics, School of Physical Sciences, Shahjalal University of Science and Technology, Sylhet-3114, Bangladesh

noraida@usm.my, mdkamal@uum.edu.my, ruzelan@uum.edu.my, lakawsar@yahoo.com, anton@usm.my, adli@cs.usm.my

Abstract: An *M/G/C/C* queuing network is a useful tool for modeling entity congestion. However, since its service rates is dependent on the number of entities (e.g. pedestrians, vehicles, etc.) in its system, direct modeling of such dynamic rates is difficult to be implemented in any modern Discrete Simulation System (DES) software. We designed an approach to cater this constraint and implemented it in Arena software to evaluate various performances of the *M/G/C/C* network. Using the models, we have compared their results with analytical results and the results of Cruz, Smith and Medeiros (2005)'s simulation models on the impacts of various arrival rates to the throughput, the blocking probability, the expected service time and the expected number of entities in a single and complex network topologies. Results indicated that there are very small discrepancies between the two DES models. Detail results on how close our simulation results and theirs have been documented and discussed.

Key words: Discrete-event simulation, *M/G/C/C* queuing systems, finite capacity, state dependent.

Introduction

M/G/C/C state dependent queuing networks are useful tools for modeling congestion systems, e.g. pedestrians in a corridor, vehicles on a road, etc. The mathematical background of the systems' behaviors and computation of their common performance measures, etc. have been discussed in details in other papers (e.g. Cruz et al., 2005; Mitchell & MacGregor Smith, 2001; Smith, 2012; Smith & Kerbache, 2012). However, since the systems treat service rates as a function of the number of their residing entities, modeling this condition is difficult in any modern Discrete Event Simulation (DES) software, .e.g. Arena (Altiok & Melamed, 2007; Kelton, 2009; Rossetti, 2010).

Cruz, Smith and Medeiros (2005) designed and constructed an *M/G/C/C* simulation model using C++ (Garrido, 1998; Prata, 2011; Watkins, 1994). The model's essential data structures, algorithms, library structures, etc. have been presented and discussed to support both the basic DES structures (Banks, Carson, Nelson, & Nicol, 2010; Wainer & Mosterman, 2010) and to cater service rates of an *M/G/C/C* network. However, its extension to support more complex or customized congestion models is probably difficult especially for model builders that do not have good experiences in programming; and this may limit its use in modeling congestion systems.

We otherwise used Arena to model *M/G/C/C* networks since it provides many DES facilities (e.g. various simulation modules, animation, statistical instrumentation and reports, etc.). However, since it does not support dynamic updating of servers' service times, we have to design a relevant modeling approach. Thus, this paper presents an approach to handle the problem and discusses how it can be implemented in Arena. Using the models, we then compare their outputs with analytical results and the results of Cruz, Smith and Medeiros's simulation models and document the findings.

We organized this paper as follows. The next Section briefly discusses the main limitation of commercial simulation software in modeling *M/G/C/C* networks and presents ideas how this limitation can be tackled. Further, we focus on the modeling of the networks using modules available in Arena software. In the following section we compare and discuss our simulation results with analytical results and the results of Cruz, Smith and Medeiros's simulation models on a single network and selected topological networks. Finally, the last section summarizes the findings and concludes the paper.

Materials and Method

Designing an *M/G/C/C* Simulation Engine

The essence of *M/G/C/C* state dependent queuing networks is that their service rates are controlled by the number of their current residing entities. Since updating a server's service time is not permitted in the current DES software, we can alternatively use a queue with a relevant buffer size that represents the maximum number of entities in a system and measure the time spent by them in the queue. Any *arrival* or *departure* event to the queue will update all entities' time spent and consequently update their remaining times to leave the queue (that is the remaining distance to cross the space).

To implement this logic, simulation software must have modules for retrieving entities from a queue so that their remaining times can be updated and modules for delaying their delay times so that they can be released whenever their remaining times become zero. Using this logic, the performances of the network can be measured using the relationships listed in Table 1.

Table 1: Model's Variables

$p(c)$ = sumBlockedPedestrians / sumArrivalPedestrians
Θ = sumDepartedPedestrians / simulationLength
L = sumTimeSpentInCorridorbyAllPedestrians / simulationLength
W = sumTimeSpentInCorridorbyAllPedestrians / sumDeparture

Arena as an Implementation Tool

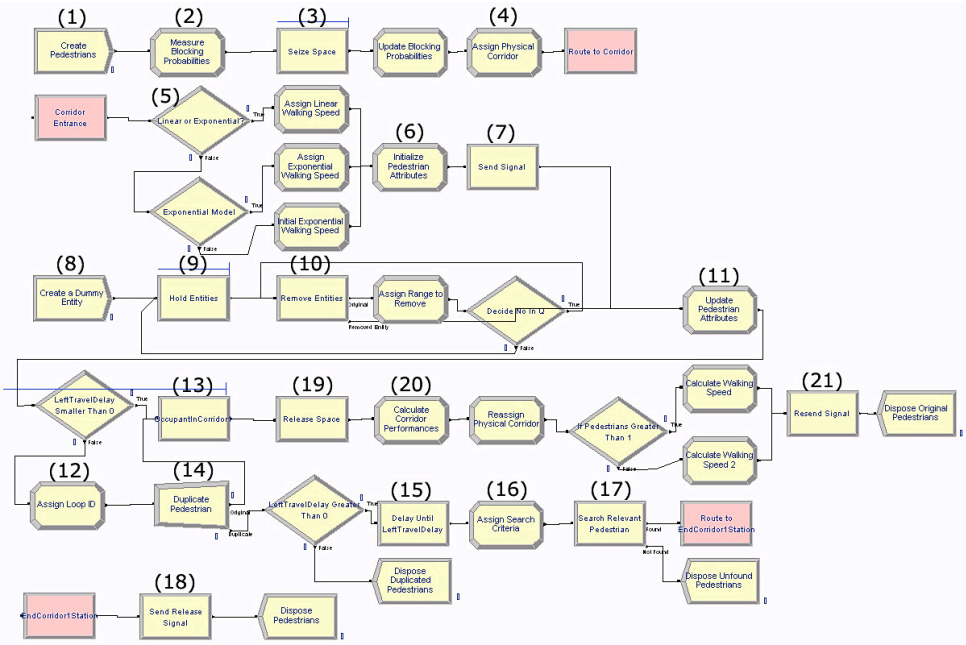

Figure 1: Arena Model for a Single *M/G/C/C* Network

Figure 1 shows our Arena model for a single *M/G/C/C* network. The model starts with the *Create* module (1) to generate a number of entities based on a certain λ arrival rate. We also have to create a dummy entity (the *Create* module (8)) to iteratively activate a mechanism to remove entities from their queue and update their current states. In the *Assign* module (2), we store the number of entering and blocking entities for calculating the network's blocking probability and declare their identification numbers (IDs) for later use in the model.

The *Seize* module (3) allocates a unit of available servers to the entity. If all available units are busy, the entity will automatically be queued until the unit is available to be seized. The *Assign* module (4) defines variables relating to the network that is its area, its capacity and its number of residing entities. These variables are used to calculate the entities' current travel speed. Since there are two mathematical models for modeling the speed (that is linear and exponential models), the *Decide* module (5) offers the option.

The *Assign* module (6) initializes entities' entrance time and current travel distance. Simultaneously, the *Signal* module (7) sends a signal (indicating an arrival event) to force the *Hold* module (9) (that is a type of queue that releases its queued entities when receiving a signal or satisfying a condition) to release the dummy entity and then activate the *Remove* module (10). The *Remove* module then removes entities from their queue (the *Queue* module (13)) to update (the *Assign* module (11)) their current travel distance, their remaining time to exit the corridor and the time points that these events happen. This time will later be used for calculating the entities' new states. We need to update (the *Assign* module (12)) their current number of state changes (loop IDs) to be used later as a search criterion in the model. The dummy entity then flows back to the *Hold* module (9) and waits for the next signal.

The entities cannot calculate their remaining times to stay while in queue. Thus, they have to be duplicated using the *Separate* module (14). The original entities flow back to their queue after updating their states, while their clones perform delay time using the *Delay* module (15). The clones later enter the *Assign* module (16) where the values of their IDs and loop IDs are assigned to new variables and used as a search criteria (accomplished by the *Search* module (17)) to match their original entities that satisfy both values.

The result of the search is either true (found) or false (not found). If the original entity is not found, the clone entity will instantly be destroyed. Else, it will send a signal (the *Signal* module (18)) to the *Hold* module (13) that will release the satisfying entity from its queue and then release (the *Release* module (19)) the network's space. Before destroyed, the entity measures the performances of the network using the *Assign* module (20) and sends a signal (the *Signal* module (21)) to the *Hold* module (13) to force all other entities to update their new states.

The basic model can easily be extended to support *series*, *splitting* and *merging* topologies. The trick is to provide relevant logics to flow entities from corridor to corridor. First, we have to create a unique queue for each corridor so that we can store its residing pedestrians and update them accordingly whenever there is an arrival or a departure event. Second, we have to attach an attribute to the entities (e.g. *toCorridor* that will take their next corridor number) so that we can travel them correctly from corridor to corridor. The value of the attribute must be updated once a relevant pedestrian exits its current corridor and used throughout the model to support the logical statements of the model, e.g. when we want to remove or search relevant entities in their queues. Third, we have to create and send a unique signal number every time the entity enters/exits their network to enable us to update their current states in the network.

Results and Discussion

All results of our simulation models were run for 20000 seconds and 30 replications. The results for a single node's performance measures are presented in Table 2 and Table 3. Table 2 displays results for a corridor 8 meters long and 2.5 meters wide under various arrival rates while Table 3 reports results for a corridor 8 meters long with different width and arrival rates. Note that Simulation[a] denotes Cruz, Smith and Medeiros (2005)'s simulation model while Simulation[b] denotes our simulation model. λ, *p(c)*, *Θ, L* and *W* respectively represent arrival rate, throughput, blocking probability, the expected service time and the expected number of entities of the network.

From Table 2, we can clearly observe that the increase of arrival rates will increase the blocking probabilities, and thus decrease the throughput. Both models display almost the same results unless for λ = 2.7 ped/s where our simulation model reports higher blocking probability compared to the analytical result and the result of Cruz, Smith and Medeiros (2005)'s simulation model. Table 3 also reports almost similar simulation results unless for a corridor width 4.5 meters with λ = 5 ped/s. In this case, our model reports lower blocking probability and this value is close

to the analytical result with 0.00 blocking probability.

Table 2: Single Node Performance Measures Versus Arrival Rate

λ	Model	*p(c)*	*Θ*	*L*	*W*
1.0	**Analytical**	0.00	1.00	6.02	6.02
	Simulation[a]	0.00 [0.00, 0.00]	1.00 [1.00, 1.00]	6.02 [5.99, 6.02]	6.02 [6.02, 6.02]
	Simulation[b]	0.00 [0.00, 0.00]	1.00 [1.00, 1.00]	6.02 [6.02, 6.04]	6.02 [6.02, 6.02]
2.0	**Analytical**	0.00	2.00	14.49	7.24
	Simulation[a]	0.00 [0.00, 0.00]	2.00 [1.99, 2.00]	14.46 [14.42,14.49]	7.24 [7.23, 7.25]
	Simulation[b]	0.00 [0.00, 0.00]	2.00 [1.99, 2.00]	14.46 [14.43,14.49]	7.24 [7.23, 7.25]
2.7	**Analytical**	0.01	2.66	29.90	10.98
	Simulation[a]	0.01 [0.00, 0.02]	2.67 [2.65, 2.68]	27.91 [26.28,29.55]	10.49 [9.81, 11.18]
	Simulation[b]	0.06 [0.03, 0.10]	2.53 [2.43, 2.62]	41.59 [32.29,50.89]	17.76 [12.76,22.75]
3.0	**Analytical**	0.33	2.01	96.96	48.31
	Simulation[a]	0.32 [0.32, 0.33]	2.03 [2.02, 2.04]	95.21 [94.52,95.90]	46.95 [46.37,47.53]
	Simulation[b]	0.34 [0.34, 0.35]	1.97 [1.96, 1.98]	97.44 [96.75,98.14]	49.57 [48.96,50.19]
4.0	**Analytical**	0.51	1.96	99.01	50.53
	Simulation[a]	0.51 [0.51, 0.51]	1.96 [1.96, 1.96]	98.77 [98.75,98.79]	50.43 [50.41,50.45]
	Simulation[b]	0.52 [0.52, 0.52]	1.93 [1.93, 1.93]	99.76 [99.75,99.77]	51.66 [51.65,51.67]

Table 3: Single Node Performance Measures Versus Width

λ	Width	Model	*p(c)*	*Θ*	*L*	*W*
2.5	**1.0**	**Analytical**	0.68	0.79	39.53	50.02
		Simulation[a]	0.68 [0.68, 0.68]	0.79 [0.79, 0.79]	39.46 [39.45, 39.46]	49.98 [49.96, 49.99]
		Simulation[b]	0.69 [0.69, 0.69]	0.78 [0.78, 0.78]	39.93 [39.93, 39.93]	51.42 [51.42, 51.43]
	1.5	**Analytical**	0.52	1.19	59.05	49.69
		Simulation[a]	0.52 [0.52, 0.52]	1.19 [1.19, 1.19]	58.91 [58.90,58.93]	49.59 [49.57, 49.61]
		Simulation[b]	0.53 [0.53, 0.54]	1.16 [1.16, 1.16]	59.87 [59.06, 59.87]	51.55 [51.54, 51.56]
	2.0	**Analytical**	0.36	1.61	77.71	48.32
		Simulation[a]	0.35 [0.35, 0.35]	1.62 [1.61, 1.62]	76.87 [76.63, 77.11]	47.50 [47.23, 47.76]
		Simulation[b]	0.38 [0.37, 0.38]	1.55 [1.55, 1.56]	79.28 [79.10, 79.45]	50.99 [50.79, 51.18]
	2.5	**Analytical**	0.00	2.50	21.07	8.43
		Simulation[a]	0.00 [0.00, 0.00]	2.50 [2.49, 2.50]	21.00 [20.94, 21.06]	8.41 [8.40, 8.43]
		Simulation[b]	0.00 [0.00, 0.00]	2.49 [2.47, 2.51]	22.71 [19.41, 26.01]	9.23 [7.60, 10.85]
	3.0	**Analytical**	0.00	2.50	18.39	7.36
		Simulation[a]	0.00 [0.00, 0.00]	2.50 [2.49, 2.50]	18.35 [18.31, 18.39]	7.35 [7.34, 7.36]

		Simulation[b]	0.00 [0.00, 0.00]	2.50 [2.49, 2.50]	18.38 [18.33, 18.44]	7.36 [7.35, 7.36]
5.0	**2.0**	**Analytical**	0.69	1.56	79.54	51.00
		Simulation[a]	0.69 [0.69, 0.69]	1.56 [1.56, 1.56]	79.40 [79.39, 79.40]	50.95 [50.95, 50.96]
		Simulation[b]	0.69 [0.69, 0.69]	1.55 [1.55, 1.55]	79.86 [79.85, 79.86]	51.67 [51.67, 51.67]
	3.0	**Analytical**	0.53	2.34	119.10	50.86
		Simulation[a]	0.53 [0.53, 0.53]	2.34 [2.34, 2.34]	118.83 [118.81, 118.84]	50.76 [50.75, 50.77]
		Simulation[b]	0.54 [0.54, 0.54]	2.32 [2.32, 2.32]	119.72 [119.72, 119.73]	51.70 [51.69, 51.71]
	4.0	**Analytical**	0.37	3.14	158.24	50.46
		Simulation[a]	0.37 [0.37, 0.37]	3.17 [3.16, 3.17]	156.04 [155.55, 156.53]	49.29 [49.01, 49.57]
		Simulation[b]	0.38 [0.37, 0.38]	3.12 [3.11, 3.13]	157.91 [157.35, 158.47]	50.69 [50.36, 51.03]
	4.5	**Analytical**	0.11	4.45	95.66	21.49
		Simulation[a]	0.00 [0.00, 0.00]	4.99 [4.97, 5.00]	46.80 [45.54, 48.05]	9.39 [9.10, 9.68]
		Simulation[b]	0.04 [0.00, 0.07]	4.99 [4.99, 5.00]	61.67 [48.36, 74.98]	13.66 [9.70, 17.62]
	5.0	**Analytical**	0.00	5.00	40.94	8.19
		Simulation[a]	0.00 [0.00, 0.00]	5.00 [4.99, 5.00]	40.88 [40.80, 40.96]	8.18 [8.18, 8.19]
		Simulation[b]	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]	40.88 [40.80, 40.96]	8.18 [8.18, 8.19]

Table 4, Table 5 and Table 6 respectively report the results for series, splitting and merging network topologies. For both series and splitting topologies, our model measures higher blocking probabilities for the first node compared to analytical and Cruz, Smith and Medeiros (2005)'s results. As a result, our model reports slightly lower throughputs for other consequence nodes. For the merging corridor, our model shows totally different blocking probability that is 0.01 compared to 0.51 (analytical result) and 0.50 (Cruz, Smith and Medeiros (2005)'s result). We believe that our model reports the right blocking probability if we follow the mathematical equation for measuring the throughput that is $\Theta=\lambda(1-p(c))$.

Table 4: Results for 3-Node Series Topology (λ = 3.0)

Measure	**Node 1**		**Node 2**		**Node 3**	
	Analytical	**Simulation[a], Simulation[b]**	**Analytical**	**Simulation[a], Simulation[b]**	**Analytical**	**Simulation[a], Simulation[b]**
p(c)	0.33	0.33 [0.32, 0.33]	0.00	0.01 [0.00, 0.01]	0.00	0.01 [0.00, 0.03]
		0.35 [0.34, 0.35]		0.00 [0.00, 0.00]		0.00 [0.00, 0.00]
Θ	2.01	2.02 [2.01, 2.03]	2.01	2.02 [2.01, 2.02]	2.01	2.02 [2.01, 2.02]
		1.96 [1.95, 1.96]		1.96 [1.95, 1.96]		1.96 [1.95, 1.96]
L	96.96	95.87 [95.35, 96.40]	14.56	16.44 [15.00, 17.88]	16.51	16.51 [15.23, 17.79]
		98.10 [97.61, 98.59]		[14.16, 14.52]		14.33 [14.15, 14.52]
W	48.31	47.53 [47.10, 47.96]	7.26	8.15 [7.44, 8.86]	8.19	8.19 [7.56, 8.81]

		50.16 [49.73, 50.59]		7.33 [7.26, 7.41]		7.33 [7.26, 7.11]

Table 5: Results for 3-Node Split Topology ($\lambda = 3.0$)

Measure	Node 1		Node 2		Node 3	
	Analytical	Simulation[a], Simulation[b]	Analytical	Simulation[a], Simulation[b]	Analytical	Simulation[a], Simulation[b]
p(c)	0.33	0.32 [0.32, 0.33]	0.00	0.00 [0.00, 0.00]	0.00	0.00 [0.00, 0.00]
		0.35 [0.34, 0.35]		0.00 [0.00, 0.00]		0.00 [0.00, 0.00]
Θ	2.01	2.03 [2.02, 2.04]	1.20	1.22 [1.21, 1.23]	0.80	0.81 [0.81, 0.82]
		1.96 [1.95, 1.97]		1.18 [1.17, 1.18]		0.78 [0.78, 0.79]
L	96.96	95.04 [94.23, 95.84]	7.48	7.61 [7.55, 7.67]	4.70	4.76 [4.73, 4.80]
		98.04 [97.38,98.71]		7.30 [7.24, 7.36]		4.58 [4.54, 4.61]
W	48.31	46.82 [46.16, 47.49]	6.21	6.24 [6.23, 6.26]	5.86	5.87 [5.86, 5.88]
		50.12 [49.54, 50.70]		6.21 [6.19, 6.23]		5.85 [5.84, 5.85]

Table 6: Results for 3-Node Merge Topology ($\lambda = \lambda_2 = 3.0$)

Measure	Node 1		Node 2		Node 3	
	Analytical	Simulation[a], Simulation[b]	Analytical	Simulation[a], Simulation[b]	Analytical	Simulation[a], Simulation[b]
p(c)	0.67	0.6 [0.68, 0.68]	0.67	0.68 [0.68, 0.68]	0.51	0.50 [0.50, 0.50]
		0.67 [0.67, 0.68]		0.68 [0.67, 0.69]		0.01 [0.01, 0.01]
Θ	0.98	0.97 [0.97, 0.97]	0.98	0.97 [0.97, 0.97]	1.96	1.93 [1.93, 1.93]
		0.98 [0.96, 1.00]		0.96 [0.93, 0.98]		1.93 [1.93, 1.93]
L	99.51	98.63 [98.60, 98.67]	99.51	98.61 [98.56, 98.67]	99.02	99.76 [99.75, 99.76]
		99.54 [99.48, 99.59]		99.55 [99.19, 99.92]		99.77 [99.76, 99.77]
W	101.6	102.0 [101.8, 102.2]	101.6	101.9 [101.8, 102.1]	50.54	51.70 [51.70, 51.71]
		101.96 [99.76, 104.16]		104.56 [102.25, 106.88]		51.71 [51.70, 51.71]

Conclusions

We managed to design and construct *M/G/C/C* state dependent simulation models using Arena software. We believe it can easily be extended by modelers that are familiar with building DES models using module approaches. We have also presented a comprehensive comparison of the results of single, series and splitting topologies of two *M/G/C/C* simulation models. The results indicated that there are very small discrepancies between the two DES models.

Acknowledgements

This study was supported by the Research University (RU) Grant Scheme, [account number 1001/PJJAUH/811097], Universiti Sains Malaysia. L. A. Kawsar wishes to thank Universiti Sains Malaysia for the financial support (USM Fellowship). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

Altiok, T., & Melamed, B. (2007). *Simulation Modeling and Analysis with ARENA*. Burlington: Academic Press.

Banks, J., Carson, J.S., Nelson, B.L., & Nicol, D.M. (2010). *Discrete-Event System Simulation* (5 edition ed.). New Jersey: Pearson.

Cruz, F.R.B., Smith, J.M., & Medeiros, R.O. (2005). An M/G/C/C State Dependent Network Simulation Model. *Computers and Operations Research, 32*(4), 919 - 941.

Garrido, J. (1998). *Practical Process Simulation Using Object-Oriented Techniques and C++* Norwood: Artech House.

Kelton, W.D. (2009). *Simulation with Arena* (5th ed.). New York: McGraw-Hill.

Mitchell, D.H., & MacGregor Smith, J. (2001). Topological network design of pedestrian networks. *Transportation Research Part B: Methodological, 35*(2), 107-135.

Prata, S. (2011). *C++ Primer Plus* (6th ed.). Indiana: Addison-Wesley Professional.

Rossetti, M.D. (2010). *Simulation Modeling and Arena*. New Jersey: John Wiley & Sons.

Smith, J.M. (2012). Topological arrangements of M/G/c/K, M/G/c/c queues in transportation and material handling systems. *Computers & OR, 39*(11), 2800-2819.

Smith, J.M., & Kerbache, L. (2012). State Dependent Models of Material Handling Systems in Closed Queueing Networks. *International Journal of Production Research*.

Wainer, G.A., & Mosterman, P.J. (2010). *Discrete-Event Modeling and Simulation: Theory and Applications (Computational Analysis, Synthesis, and Design of Dynamic Systems)*. Boca Raton: CRC Press.

Watkins, K. (1994). *Discrete Event Simulation in C*. New York: McGraw-Hill Companies.