Journal of the Operations Research
Society of Japan
Vol. 29, No. 2, June 1986

A FLUID FLOW APPROXIMATION ANALYSER FOR BUFFER TYPE QUEUEING SYSTEMS

I. M. Premachandra
University of Sri Lanka

Hidenori Morimura
Tokyo Institute of Technology

(Received November 1, 1985: Final February 17, 1986)

Abstract Queueing systems in which idle times of servers seldom occur are commonly seen in many practical areas such as manufacturing, transportation, etc. Here we call such a system buffer type queueing system. It is known that the fluid flow approximation technique is efficient in analysing such systems. This paper presents an algorithm based on this technique and a comprehensive computing tool (FFQA) which employs it. The applicability and the accuracy of FFQA are illustrated through some examples.

1. Introduction

In practice, there are many examples of a type of queueing system in which idling of the servers seldom occurs. Let us consider a situation of unloading ores from a freighter to barges. As far as there are empty barges near the freighter, the unloading process continues. In case of manufacturing processes, they usually have no interruption due to the shortage of materials since buffer storage is available. In the present paper, we shall call such queueing systems buffer type.

Newell, G.F.[6] proposed the so-called fluid flow approximation technique to analyse such queueing systems. It is known that the technique is efficient and simple. Also, Nakamura, Z.[5] recommends to use it in analysing work system improvement. But, when we apply the technique to analysis of a complex queueing system such as above examples, it is not so simple to obtain some evaluation quantity (e.g., mean waiting time) analytically. We analysed such a case[11]. If we have a computing tool that can be manipulated easily, then such an analysis becomes simple.

The purpose of this paper is firstly, to develop an algorithm based on the fluid flow approximation to compute the mean sojourn time or the mean number of customers in complex queueing systems and to test whether it could be used efficiently with a sufficient accuracy. The second purpose is to illustrate the outline and the computational experience of a program package FFQA(Fluid Flow Queueing Analyser) which employs this algorithm.

When we have no any computing tool, we may use some popular simulation language such as GPSS[12], Q-GERT[2], SLAM II[1,13,14] etc. If we use them, of course we can obtain many informations about the system behaviour, but the treatment is not so simple. We are expected to be familiar with many inherent terminologies and symbols.

The rest of the paper is organized as follows. Section 2 discusses the basic ideas behind the algorithm. Section 3 illustrates the outline of FFQA. Some numerical examples are given in § 4 and a comparative study of FFQA with other simulation languages appears in § 5. Finally, Section 6 presents some concluding remarks.

2. Basic Ideas Behind the Algorithm

We summarize below the basic ideas behind the algorithm; available in Appendices A and B.

2.1 Concept of Node

The flow of different commodities in a system is taken into consideration by defining service stations with respect to each commodity. We call such service stations nodes in order to distinguish them from service stations in usual sense. Hence, if there is only one type of commodity flowing through the system then the nodes mean the existing service stations. To explain this concept consider the following example.

Example - 1 (loading unloading problem)

Consider the loading and unloading problem of an ore freighter. Raw materials brought by the freighter are first loaded into the barges. The loaded barge is then pulled to the yard by a tugboat. At the yard the loaded barge is unloaded by a crane. The empty barges generated at the yard due to unloading are pulled back to the freighter in order to keep the process going. This loading unloading process is continued until the materials in the freighter are completely unloaded.

In usual terminologies, the first service station in this process is an unloading crane on the freighter. Input customers are ores and output customers are loaded barges. The server is interrupted if there are no empty barges. The server in the second service station is a tugboat and the customers are loaded barges. This approach is rather complex But, if we consider the following seven types of customers and appropriate seven nodes, we can explain the system in a natural form as illustrated in Fig. 1

i) Materials in the freighter (MAT)
ii) Empty barges waiting near the freighter (EBF)
iii) Tugboats waiting near the freighter (TAF)
iv) Loaded barges waiting near the freighter (LBF)
v) Loaded barges waiting at the yard (LBY)
vi) Empty barges waiting at the yard (EBY)
vii) Tugboats waiting at the yard (TAY).

For the notational convenience hereafter we shall denote a node only by a circle with a number inside.

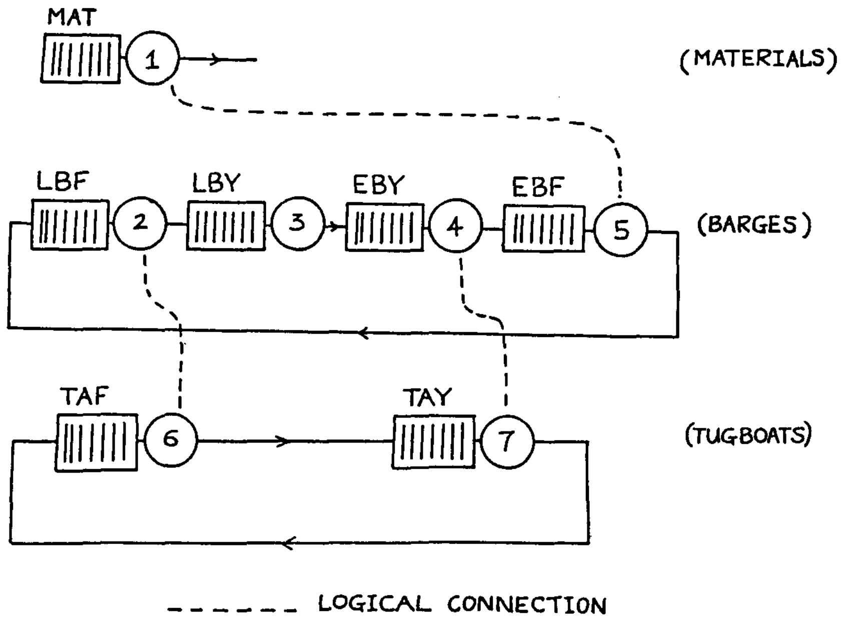

Fig. 1 Flow Diagram of the Loading Unloading Problem

2.2 Fluid Flow Approximation

Under the heavy traffic condition, we can use the fluid flow approximation method to approximate the queueing processes.

When the nodes are combined into a network form such as in Fig. 1, the output from one node is considered to be the input to the next immediate nodes in the flow diagram. The direction of the flow of commodities in the flow diagram is indicated by an arrow.

2.3 Constraint and Induced Delay

The flow diagram itself may not give a full description of the real system under consideration unless the interdependence of the nodes is defined.

Thus, with respect to each node i of the flow diagram we have to define (if necessary) a set of constraints in order to detect the pattern of the departure process from the node If at least one of these constraints is satisfied at any time t, then FFQA sets the departure process from node i to an interrupted state for a certain time interval which we call here induced delay. The following example illustrates this fact and for details refer [8,9,10].

Example - 2 (tandem queue with blocking)

Consider a two stage tandem queue which serves a single type of commodity. Assume that the node 2 (i.e., stage 2) has a finite waiting room of capacity 3. The departing commodity from node 1 is assumed to be blocked if the waiting room at the second node is full.

Let $A_i(t)$ and $D_i(t)$ be the cumulative arrival to node i and the cumulative departure from node i respectively.

In this case we have to define the condition $A_2(t) - D_2(t) = 3$ as a constraint to node 1. Whenever (time t_0 in Fig. 2) this constraint is satisfied, FFQA interrupts the departure process from node 1 for a time interval μ_{d2}^{-1}, where μ_{d2} is the service rate at node 2. The dotted lines in Fig. 2 indicate the process after the induced delay.

2.4 Additional Node

In modeling a real system using FFQA, one may require to model a situation where a commodity takes a certain time DT_i to move between two nodes i and j. In this case we have to input only the delay DT_i so that FFQA automatically introduces an additional node into the flow diagram between the nodes i and j. The mean service time of this node is considered to be DT_i. Strictly speaking the arrival and the departure processes of the additional node r can be given as $A_r(t) = D_i(t)$ and $D_r(t) = A_r(t+DT_i)$, where r is the number assigned to the additional node by FFQA.

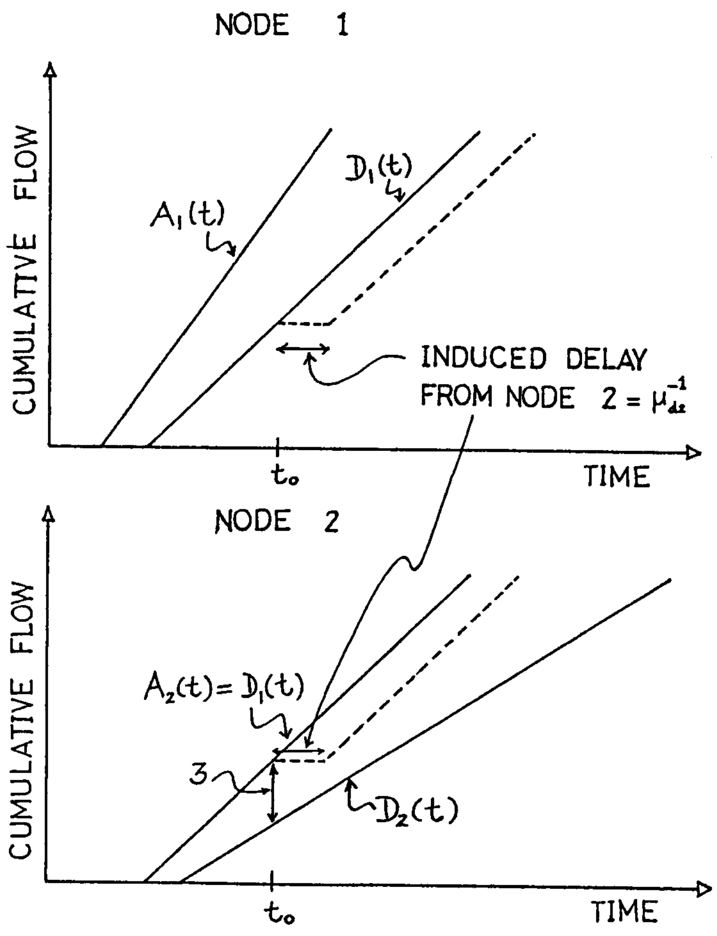

Fig. 2 Induced Delay in Case of Blocking

2.5 Indicator Variable

FFQA checks whether the constraints defined to each node are satisfied or not on the basis of the indicator variables defined as follows.

$$CC(i,j) = \begin{cases} 1 & \text{if the j-th constraint of node i is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1,2, \ldots, M$ and $j = 1,2, \ldots n_i$ where M is the total number of nodes and n_i is the number of constraints defined to node i. So, if at any time t, the value $\sum_{j=1}^{n_i} CC(i,j) \neq 0$ then FFQA sets the departure process from node i to an interrupted state. Refer Appendices A and B for details.

2.6 Termination of the Computation

The ending of the computation in FFQA is determined by means of a termination condition defined by the user. Informations such as the time at which the computational process is to be stopped or the number of commodities required to be processed at a particular node before the termination can be used to define the termination condition. A detailed discussion is available in [10].

2.7 A Rapid Algorithm

Appendix A discusses the main algorithm used in FFQA. In this algorithm, the computation is performed by incrementing the simulation time in small time intervals. We call the software associated with this algorithm FFQA-GAP (FFQA - General APproach). This approach may allow the user to input any type of constraint in BASIC statements and therefore a wide variety of queueing systems including networks can be tackled.

But on the other hand, the algorithm in FFQA-GAP may increase the CPU time because the clock time is incremented by small time intervals. Therefore we propose a rapid algorithm (see Appendix B) and its program package FFQA-RAPID, by removing the condition that the clock time is incremented by small time intervals. In principle, the computational process in FFQA-RAPID searches the minimum time point t^* of t_i^* at which $\sum_{j=1}^{R_i} CC(i,j) \neq 0$ for each node i and advances the simulation time to t^*(see Appendix B). Here R_i is the number of constraints that uses informations about the node i. In order to ease the computation in FFQA-RAPID we impose here the following restrictions.

(a) Restrictions imposed on the flow diagram

i) Flow diagram should have only one initial node. Here an initial node is a node which does not have incoming arrows which are emanated from other nodes; e.g., node 1 in Fig. 1.

ii) The input set and the output set of all the nodes should consist of at most one element. Here the input set of any node i is the set of nodes whose outgoing arrow becomes an incoming arrow to node i. Similar definition can be given to the output set.

(b) Restrictions imposed on the constraints

FFQA-RAPID allows only the constraints of the form

$$A_i(t) - D_i(t) = p \quad (p \geqq 0) \text{ or}$$
$$A_i(t) - D_i(t) \geqq q \quad (q > 0).$$

Note that these constraints are the types that are frequently used in modeling many of the queueing situations using FFQA. Because of the above restrictions the applicability range of FFQA-RAPID becomes narrow compared with FFQA-GAP. It is also to be noted that in future it may be possible to release the restriction (a).

3. Outline of FFQA

FFQA is written in BASIC and is run on a personalcomputer (NEC PC-9801, 528 KBytes). Input data to the package is received by three subroutines DATA 1 ~ 3.

DATA 1 receives incremental simulation time Δ (=0.25 in the default case) and the general data corresponding to each node such as node number, service rate, queue size at t=0, input set, output set, percentage of commodities moved from node to node and the delay occured in moving a commodity from one node to another. Statements such as NODE NUMBER: SERVICE RATE etc., will appear on the screen automatically so that the user can input the relevant data through the keyboard.

In DATA 2, FFQA checks the initial nodes in the flow diagram on the basis of the data in DATA 1 and will request the user to input the data such as arrival rate, time arrival begins etc., corresponding to them. Various arrival patterns will appear on the screen and the user can input these values through the keyboard.

DATA 3 is for inputting the informations such as constraints, termination condition etc., in BASIC statements. In principle, any appropriate mathematical expression is permitted as a constraint.

Summary report (see [10]) prints out all the input data and the output data such as mean sojourn time, cumulative arrival, cumulative departure, maximum, minimum and the average number of commodities with respect to each node.

Due to the conversational ability of FFQA, the manipulation is very simple.

4. Numerical Examples

In order to illustrate the applicability of FFQA-GAP and FFQA-RAPID, we present here some numerical examples and for details and more examples refer [7]. The accuracy of the results from FFQA is checked by comparing with GPDS (General Purpose Discrete Simulator - almost same as GPSS) results obtained from Melcom Cosmo 700 machine.

4.1 Tandem Queue with Blocking

Consider Example - 2 and assume that the service rates in node 1 and 2 be 0.51 and 0.32 respectively. It is also assumed that the commodities arrive at node 1 at the rate of μ_{a1} . The system is simulated using FFQA-RAPID and GPDS and the results appear in Table 1 below.

In Table 1, GPDS(det) refers to the case where the interarrival times and the service times at the nodes are constants at $1/\mu_{a1}$, $1/.51$ and $1/.32$ respectively. For the sake of comparison, we also consider an extreme case where these times are exponentially distributed with the same means as above and the corresponding results appear under the column GPDS(ran). The bracketed figure is the estimated standard deviation from 12 runs.

It can be seen from Table 1 that for values $\mu_{a1} > 0.5$ (during this range the traffic intensity of node 1 > 1)FFQA gives good approximate values. In the case of node 2, the fit is good even for $\mu_{a1} > 0.4$. results corresponding to the random case are almost the same as that in the deterministic case. It is to be noted that the same set of random numbers is used to estimate the means and the standard deviations corresponding to the different values of μ_{a1}.

μ_{a1}	Node number	Avg.Sojourn time GPDS(ran)	GPDS(det)	FFQA-RAPID	Maxi.contents GPDS (det)	FFQA-RAPID
.9	1	46.35(11.86)	45.91	46.14	30	30.1
	2	8.35(1.11)	8.40	8.28	3	3
.8	1	42.83(11.7)	42.43	42.74	27	28
	2	8.28(1.16)	8.46	8.28	3	3
.7	1	39.01(11 5)	38.2	38	25	25.3
	2	8.23(1.15)	8.51	8	3	3
.6	1	33.3 (10.8)	32.2	32.5	21	21.3
	2	8.22(1.22)	8.56	8.28	3	3
.5	1	27.0 (9.4)	23.83	24.3	16	15.9
	2	7.96(1.37)	8.67	8.28	3	3
.4	1	18.4 (7.25)	11.76	12.2	8	8.4
	2	7.71(1.3)	8.5	8.18	3	3

Table 1. Average Sojourn Times of the Tandem Queue

We also use Example - 2 to illustrate the efficiency of FFQA-RAPID over FFQA-GAP. The computed values are illustrated in Table 2 below along with the computation times.

	FFQA-RAPID	FFQA-GAP $\Delta = 0.5$	$\Delta = .25$
average sojourn time in node 1	42.74	43.12	42.85
average sojourn time in node 2	8.28	8.22	8.26
time taken to input data	1 minute & 12 seconds	3 minutes	3 minutes
computation time	2 minutes & 30 seconds	7 minutes	14minutes

Table 2. Computation Times of FFQA

Table 2 shows that FFQA-RAPID is faster than FFQA-GAP. It can also be seen that when Δ (incremental simulation time) decreases the accuracy of the computed values from FFQA-GAP increases eventhough the CPU time becomes large.

In simulating the problems hereafter, preference is given to

FFQA-RAPID due to the time factor and only the problems that cannot be tackled by FFQA-RAPID are simulated by FFQA-GAP.

4.2 Tandem Queue with a Delay

We consider here a tandem queue with 3 nodes and it is assumed that it takes 2 time units for a commodity to move from node 2 to 3. Furthermore, we assume that at time t=0 there are 20 commodities in node 1 and after that no arrival occurs. The service rates of each node are assumed to be 0.8,0.5 and 0.3 respectively.

In this case FFQA automatically introduces an additional node to the flow diagram between the nodes 2 and 3 with an average service time of 2, before the computation starts. The computed values appear in Table 3 below.

Node	Avg.sojourn time			Maxi.contents	
number	GPDS(ran)	GPDS(det)	FFQA-RAPID	GPDS	FFQA-RAPID
1	12.64(3.14)	13.12	13.12	19	20
2	10.39(5.72)	9.12	9.12	8	8.3
3	15.84(7.0)	15.96	16	8	8.8

Table 3. Average Sojourn Time of a Tandem Queue with Delay

4.3 Loading Unloading Problem

Consider Example - 1. In this case one of the objectives may be to minimize the waiting time of the freighter. So, for example one may be interested in finding out the optimal number of barges required to minimize the waiting time of the freighter under say, a cost constraint. It is assumed here that the loading rate of materials into the barges is 0.8 and the unloading rate at the yard is 0.5. It is also assumed that it takes 3 time units for the commodities to move between the nodes $2\to3$, $4\to5$, $6\to7$ and $7\to6$ in Fig. 1. For the details of the input data refer [10].

The average waiting time of the freighter is plotted in Fig. 3 against the number of barges. It can be seen from Fig. 3 that FFQA-RAPID gives close values to that of GPDS.

4.4 A Queueing System with Feed Back

Let's consider the queueing system illustrated in Fig. 4 below. Assume that the service rates of the nodes 1,2 and 3 are 0.8, 0.6and 0.4 respectively. 60% of the commodities departing from node 3 join node 2 again and the rest depart from the system. It is also assumed that the queue size in node 1 at t=0 is 50 and no arrival will occur thereafter. The system is simulated using FFQA-GAP and GPDS until 60 commodities

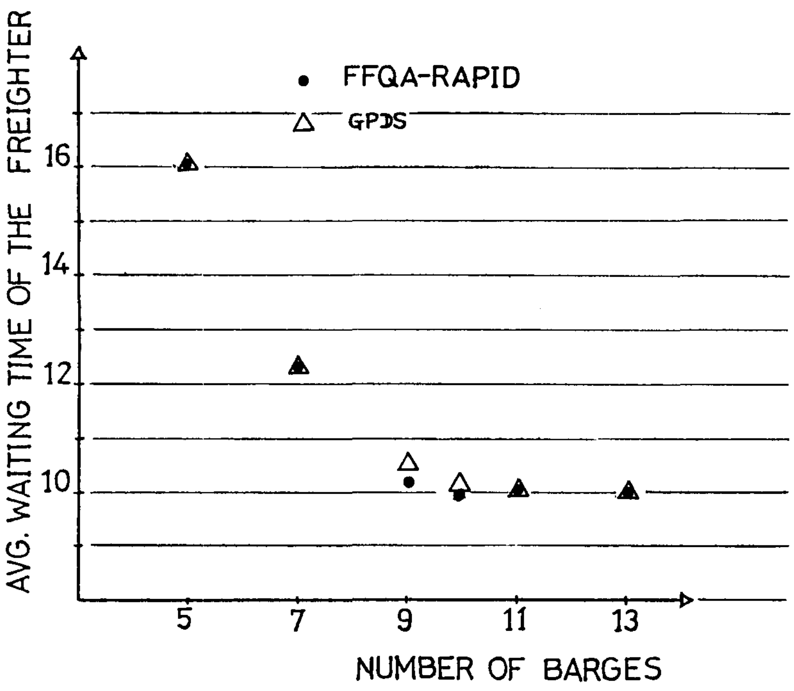

Fig. 3 Average Waiting Time of the Freighter

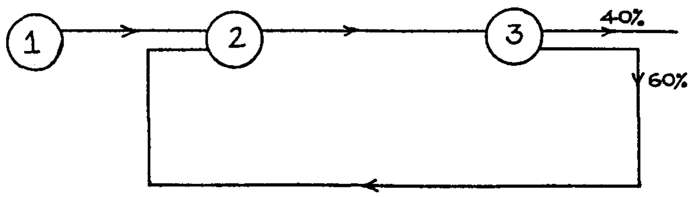

Fig. 4 Queueing System with Feed Back

are served at node 3 and the output data is listed in Table 4.

	Avg.sojourn time	
	FFQA-GAP $\Delta = .25$	GPDS(det)
average sojourn time in node 1	32.00	31.87
average sojourn time in node 2	21.93	21.89
average sojourn time in node 3	28.02	26.53

Table 4. Average Sojourn Times of a Feed Back Queue

4.5 A Complicated Network Type Queueing System

Consider the queueing network illustrated by the flow diagram in Fig. 5. The percentages of commodities moved along in each direction are indicated in the diagram. The service rates of the nodes are assumed to

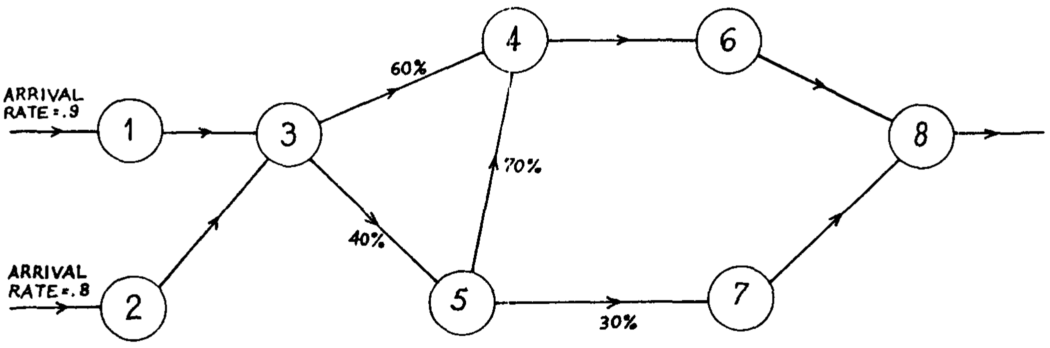

Fig. 5 Flow Diagram of the Network

be 0.4, 0.5, 0.5, 0.4, 0.4, 0.5, 0.3 and 0.5 respectively. The queue sizes of the nodes 2 ~ 7 at t=0 are assumed to be 10, 8, 12, 15, 8 and 20 respectively. This example is presented here in order to illustrate the capability of FFQA to tackle complicated network type queueing systems. The computed results from FFQA-GAP and GPDS are shown in Table 5.

Node number	Avg.sojourn time	
	FFQA-GAP $\Delta = .25$	GPDS(det)
1	34.36	34.42
2	33.78	33.20
3	28.79	29.23
5	16.99	17.15
4	31.43	33.34
7	36.53	33.33
6	6.93	7.1
8	20.01	20.8

Table 5. Average Sojourn Times of the Network

Note that the computing order of the nodes determined by the sub algorithm *ORDER in Appendix A is 1, 2, 3, 5, 4, 7, 6, 8.

5. A Comparison with Other Methods

In this Section, we compare FFQA with other accepted simulation languages such as DYNAMO and SLAM II for the purpose of illustrating the computational efficiency and the simplicity in manipulation of FFQA.

From the numerical results in Section 4 we can see that FFQA gives reasonable approximate mean values for the queueing characteristics when the system is in heavy traffic condition. Therefore, FFQA can be used as a tool to analyse such systems.

The use of GPDS and SLAM II requires a specialized knowledge about their block symbols and the programing terminologies. But, in case of FFQA the data input procedure is very simple and therefore even a beginner can use the package without such a preparatory training. The other attraction in FFQA is its conversational ability.

We can view the computational process in FFQA as a continuous simulation or a process in which the computation is performed on the basis of a fluid flow model instead of a discrete simulation. Therefore we compare the results from FFQA with that from B-DYNAMO [3] which is a micrôcomputer version of DYNAMO. We use the example discussed in Section 4.1 for comparison purposes and the corresponding results appear in

Table 6. In Table 6 μ_{a1} is the arrival rate to the node 1. From the table we can see that in case of B-DYNAMO the computation time is less

μ_{a1}	Node	Description	GPDS(det)	FFQA-RAPID	SLAM II	B-DYNAMO
	1	Avg.sojo.time	45.91	46.14	44.35	47.35
.9		Maxi.contents	30	30.1	30	33
	2	Avg.sojo.time	8.46	8.28	8.95	9.28
		Maxi.contents	3	3	3	3.2
	1	Avg.sojo.time	42.43	42.74	40.92	44.03
.8		Maxi.contents	27	28	27	30.8
	2	Avg.sojo.time	8.46	8.28	8.95	9.28
		Maxi.contents	3	3	3	3.2
	1	Avg.sojo.time	38.2	38	36.51	39.23
.7		Maxi.contents	25	25.3	25	27.5
	2	Avg.sojo.time	8.51	8	8.95	9.28
		Maxi.contents	3	3	3	3.2
	1	Avg.sojo.time	32.2	32.5	30.63	32.82
.6		Maxi.contents	21	21.3	21	23.3
	2	Avg.sojo.time	8.56	8.28	8.95	9.28
		Maxi.contents	3	3	3	3.2
	1	Avg.sojo.time	23.83	24.3	22.56	23.79
.5		Maxi.contents	16	15.9	16	17.3
	2	Avg.sojo.time	8.67	8.28	8.94	9.28
		Maxi.contents	3	3	3	3.2
	1	Avg.sojo.time	11.76	12.2	10.74	10.73
.4		Maxi.contents	8	8.4	8	8.67
	2	Avg.sojo.time	8.5	8.18	8.66	8.96
		Maxi.contents	3	3	2	3.2
Program input time by the keyboard *				1 minute & 12 sec.	5 minutes & 30 sec.	9 minutes
Computation time per one value of μ_{a1}				2 minutes & 30 sec.	11 minutes	25 seconds

* In case of FFQA no program is required and this means the data input time.

Table 6. Average Sojourn time values from different simulation languages

than FFQA. But, One of the difficulties in using B-DYNAMO when compared with FFQA is that since it is not designed for the purpose of simulating queueing systems, the user has to add subprograms written in BASIC to the main package in order to compute the required queueing characteristics. The other difficulty is that it requires a knowledge of systems dynamics concepts [4].

Finally, the results from FFQA are compared with SLAM II which 'is considered to be a discrete simulator designed for the use in microcomputers. In this case, the computation time is larger than that in FFQA.

6. Conclusions

In this paper we proposed a simple software package based on fluid flow approximation technique to analyse buffer type queueing systems approximately. Queueing situations in heavy traffic condition are often seen in many practical areas. An engineer, however, in attacking such a practical problem may require to make some preliminary estimates in order to decide whether a system will work or not. If the decision is still in doubt after the crude estimates, then he may need more accurate estimates. Therefore, a simple tool such as FFQA that could be used to analyse such systems approximately is of practical value.

It can be seen from Table 1 ~ 6 that FFQA gives reasonable approximate values. In addition to the rapidness of computation and its accuracy, FFQA is simple to handle. Its conversational ability is another advantage. Note that FFQA can be programmed on any microcomputer such as PC-9801(528 KB) using a 8 inch or a 5 inch disk and it needs only the operating system N_{88} - BASIC.

One of the serious drawbacks of FFQA is that it cannot be used to simulate queueing systems in light traffic conditions.

Acknowledgement

The authors would like to express their sincere thanks for the referees for their valuable comments made on this paper.

Appendix A

The main algorithm mentioned in Section 2.7

We shall define here a cycle to be the fragment of the simulation process covered by a time interval Δ (= incremental simulation time).

Let CUMA(i,1) and CUMA(i,2) be the cumulative arrival to node i during the previous cycle and the current cycle respectively. CUMD(i,1) and CUMD(i,2) are the corresponding values of the cumulative departure from node i.

STEP 0 : Set t = 0. Set CUMA(i,1) = $Q_i(0)$ and CUMD(i,1) = 0 for i = 1,2,.... M, where $Q_i(0)$ is the queue size of node i at t = 0 and M is the total number of nodes. Determine the computation order of the nodes by using the sub algorithm *ORDER (see below)

STEP 1 : Increase t by Δ and calculate the cumulative arrival CUMA(i,2) and the cumulative departure CUMD(i,2) corresponding to each node i as follows. This computation is done in the order determined by the sub algorithm *ORDER.

(i) for any initial node i, CUMA(i,2) = CUMA(i,1) + $\Delta\ \mu_{ai}$ where μ_{ai} is the arrival rate to the node i.

(ii) if we let I(i) be the input set of any node i (i is not an initial node) then

$$\text{CUMA}(i,2) = \text{CUMA}(i,1) + \sum_{k\in I(i)} B(k)\ \mu_{dk} P_{ki}\ \Delta$$

where

$$B(k) = \begin{cases} 1 & \text{if } C(k,1) = 0 \\ 0 & \text{if } C(k,1) \neq 0, \end{cases}$$

μ_{dk} is the service rate of node k, P_{ki} is the fraction of commodities input from node k to i and C(k,1) is the value $\sum_{j=1}^{n_k} CC(k,j)$ corresponding to the previous cycle. Here n_k is the number of constraints defined to node k.

(iii). for any node i

$$\text{CUMD}(i,2) = \begin{cases} \text{CUMD}(i,1) & \text{if } C(k,1) \neq 0 \\ \text{CUMD}(i,1) + \mu_{di}\ \Delta & \text{if } C(k,1) = 0 \end{cases}$$

STEP 2 : calculate the areas covered by the arrival and the departure curves with the time axis during the interval Δ . Executing the BASIC statements input by the user, determine the new values of the indicator variables CC(i,j) and then calculate

$$C(i,2) = \sum_{j=1}^{n_i} CC(i,j) \text{ for } i = 1,2,\ldots,M.$$

STEP 3 : obtain the set of nodes $S = \{ h \mid C(h,2) \neq 0 \}$. With respect to the departure processes of each node in S, induce the delay corresponding to the constraint satisfied at time t.

STEP 4 : set C(i,1) = C(i,2) , CUMA(i,1) = CUMA(i,2) and CUMD(i,1) = CUMD(i,2) for i = 1,2, ... , M. If the termination condition is not satisfied then STEP 1 is executed else STEP 5.

STEP 5 : Print out the summary report

We mentioned in Section 2 that the output of any node i is considered to be the input to the next immediate nodes in the flow diagram. This implies that the computation of CUMA(i,2) and the CUMD(i,2) of the nodes should be done in a particular order. The sub algorithm *ORDER explained below is used to determine this computation order.

Sub Algorithm *ORDER

STEP 0 : Define the matrix $A = (a_{ij})$, $i = 1,2,\ldots,M$; $j = 1,2,\ldots,M$ where

$$a_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if there is an arrow emanating from node } i \text{ to } j \\ 0 & \text{if there does not exist an arrow emanating from node } i \text{ to } j \end{cases}$$

For the sake of computation, let's call a node j as an initial node if $a_{ij} = 0$ for all i. Set R = 1.

STEP 1 : Find the set of node numbers $S' = \{ k \mid a_{ik} = 0, \forall i \}$. Assign the node numbers in S' to the variables X(R), X(R+1),...., X(|S'|) in any order.

STEP 2 : For any $k \in S'$, set $a_{kj} = 0$ for $\forall$ j. Setting $a_{kj} = 0$ for $\forall$ j will create at least one more initial node. Let this set of initial nodes be S''. Assign the node numbers in S'' to the variables X(R+1),X(R+2),..., X(R+|S''|-1). This procedure is repeated with respect to each of the initial nodes so created until all the elements in A become zero.

Thus the sequence X(1),X(2),...X(M) indicates the sequential order of the nodes according to which the computation in STEP 1 of the main algorithm is done.

Appendix B

The Rapid Algorithm mentioned in Section 2.7

STEP 1

Let R_i be the number of constraints that uses informations about node i and t_{ij} be the time at which the j-th constraint that uses informations about node i is satisfied.

Consider node 1 first. We know that there are R_1 constraints that use node 1 informations. The times $t_{1j}(j=1,2,\dots,R_1)$ at which each of the R_1 constraints are satisfied are calculated and the minimum

(B.1) $t_1^* = \text{Min}(t_{11}, t_{12}, \dots, t_{1R_1})$

is obtained. The ammount that can be processed until t_1^* is input to the next node that is node 2 in Fig. 6. For node 2, the times $t_{2j}(j=1,2,\dots,R_2)$ at which each of the R_2 constraints are satisfied are calculated for an input within $t \leq t_1^*$ and their minimum

(B.2) $t_2^* = \text{Min}(t_{21}, t_{22}, \dots, t_{2R_2})$

is obtained. The same procedure is then repeated to calculate t_3^*, t_4^*, ... , t_M^*.

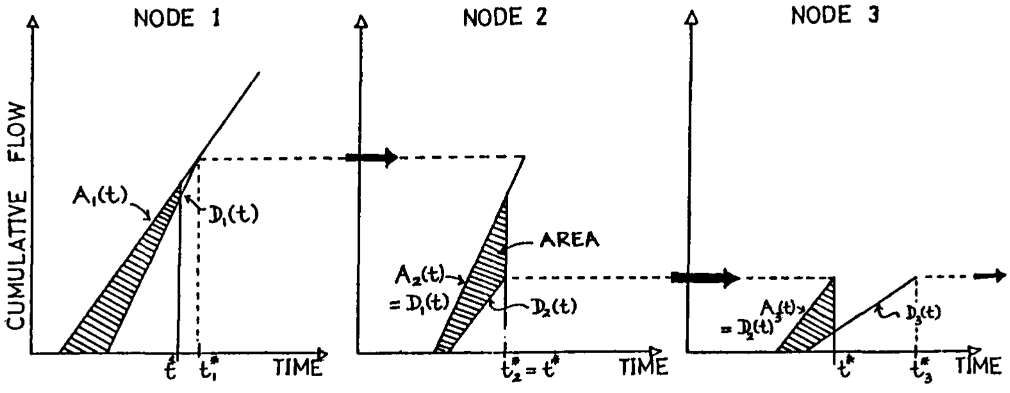

Fig. 6 Flow of Commodities among the Nodes

STEP 2

The overall minimum

(B.3) $t^* = \text{Min}\ (t_1^*, t_2^*, \ldots , t_M^*)$

is obtained and the area between the arrival and the departure processes of each node until t^* is calculated. The cumulative arrival and the cumulative departure at t^* are also calculated for each node and these informations are later used in the programme as restarting conditions..

STEP 3

For each node i(i=1,2,..., M), the quantity $\sum_{j=1}^{R_i} CC(i,j)$ is calculated and the set S of nodes whose $\sum_{j=1}^{R_i} CC(i,j) \neq 0$ is obtained. Note that for all nodes $x \in S$, $t_x^* = t^*$. For nodes $x \in S$, the induced delay corresponding to the constraint satisfied at t_x^* is induced to the departure process of the node concerned. The history of the system until t^* is neglected and the process STEP 1 STEP 3 is repeated.

References

[1] Alan B. Pritsker, A., *Introduction to Simulation and SLAM II*, John Wiley and Sons, 1984.

[2] Alan B. Pritsker, A., *Modeling and Analysis using Q-GERT Networks*, John Wiley and Sons, New York, 1977.

[3] Hayashi, S., B-DYNAMO user's Manual(Version 1.1 and 2.1), presented at the 58th Conference of the Operational Research Society of Japan, Oct 1985(in Japanese).

[4] Jay W. Forrester, *Industrial Dynamics*, M.I.T. Press, 1961

[5] Nakamura, Z., Cumulative Flow Graph Analysis for Work System Improvement, *Journal of Japan Industrial Management Association*, Vol.36 No.2, 1985 (in Japanese)

[6] Newell, G.F., *Applications of Queueing Theory*, Chapman and Hall, 1982.

[7] Premachandra, I.M., Analysis of Buffer Type Queueing Systems, Ph.D Thesis, Tokyo Institute of Technology, December 1985.

[8] Premachandra, I.M and Morimura, H., Analysis of Buffer type Queueing Systems, Research Reports on Information Sciences, No. B-169, Tokyo Institute of Technology, July 1985.

[9] Premachandra I.M and Morimura, H. Analysis of Buffer Type Queueing Systems: The General Approach, Research Reports on Information Sciences No. B-171 Tokyo Institute of Technology, September 1985.

[10] ____________, FFQA-RAPID and FFQA-GAP (manual), Research Reports on Information Sciences, No. B-170, Tokyo Institute of Technology, September 1985.

[11] ____________, Modeling and Analysis of a Queueing System Existing in a Bank, *Journal of the Operations Research Society of Japan*, Vol.28 No.2, 1985.

[12] Reference Manual - GPDS (MELCOM, NM-SR00-69A<73A0>)

[13] SLAM II PC Version User's Manual, Pritsker and Associates, Inc.

[14] SLAM II Quick Reference Manual, Pritsker and Associates, Inc.

Hidenori MORIMURA: Department of
Information Science,
Tokyo Institute of Technology,
2-12-1, Oh-okayama, Meguro-ku,
Tokyo 152, Japan.