Title: PARAMETRIC ANALYSIS OF QUEUEING NETWORKS WITH BLOCKING

PROJECT ADMINISTRATION DATA

OCA contact: Brian J. Lindberg 894-4820

Sponsor technical contact

JOHN P. THOMAS, JR., MAJOR, USAF
(202) 767-5025
AFOSR/NM
BUILDING 410
BOLLING AFB, D.C. 20332-6448

Security class (U,C,S,TS): U

Defense priority rating : N/A

Equipment title vests with: Sponsor

TITLE TO EQUIP. ACQUIRED AT A COST OF LESS THAN $5000 SHALL VEST WITH GIT

Administrative comments -
INITIATION OF PROJECT G-36-603.
NOTICE OF PROJECT CLOSEOUT

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

PROJECT: QUEUING NETWORKS WITH FINITE CAPACITIES

Effective Completion Date 891031 (Performance) 891231 (Reports)

Closeout Actions Required:

<table>
<thead>
<tr>
<th>Action</th>
<th>Y/N</th>
<th>Submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Invoice or Copy of Final Invoice</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Final Report of Inventions and/or Subcontracts</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Government Property Inventory &amp; Related Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Classified Material Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Release and Assignment</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

<table>
<thead>
<tr>
<th>Recipient</th>
<th>Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>Y</td>
</tr>
<tr>
<td>Administrative Network Representative</td>
<td>Y</td>
</tr>
<tr>
<td>GTRI Accounting/Grants and Contracts</td>
<td>Y</td>
</tr>
<tr>
<td>Procurement/Supply Services</td>
<td>Y</td>
</tr>
<tr>
<td>Research Property Management</td>
<td>Y</td>
</tr>
<tr>
<td>Research Security Services</td>
<td>N</td>
</tr>
<tr>
<td>Reports Coordinator (OCA)</td>
<td>Y</td>
</tr>
<tr>
<td>GTRC</td>
<td>Y</td>
</tr>
<tr>
<td>Project File</td>
<td>N</td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
</tr>
</tbody>
</table>

NOTE: Final Questionnaire sent to PDPI.
1. Introduction

Performance has been a major issue in the design and implementation of systems such as: computer systems, production systems, communication networks and flexible manufacturing systems. The success or failure of such systems is judged by the degree to which performance objectives are met. Thus, tools and techniques for predicting performance measures are of great interest. In the last two decades it has been demonstrated several times that performance can be evaluated and/or predicted well by queueing models which can be solved either by simulation or analytical methods. Simulation is the most general and powerful technique for studying and predicting system performance. However, the high cost of running the simulation programs and uncertain statistical accuracy, makes simulation less attractive. Compared to simulation, analytical methods are more restrictive but have the advantage that it is less costly to compute numerical results. Moreover, they can be implemented very quickly, thus it is very easy to give interpretations to the relationships between model parameters and performance measures. Analytical methods have proved invaluable in modeling a variety of computer systems, computer networks, flexible manufacturing systems, etc. They are flexible enough to represent adequately many of the features arising in such applications. They have not been able to provide much insight into the phenomenon of blocking, because all methods for networks are based on the assumption that the stations have infinite capacities. If the stations have finite capacities, blocking can occur in the network and this causes interdependencies between the stations. Hence, all the classical algorithms known in the literature cannot be applied. Therefore, in this project our aim is to solve queueing networks with finite capacities.

2. "BASIC RESEARCH" CONTRIBUTIONS

Our major contributions in this project are:
We were the first to develop very efficient computational algorithms for queueing models with finite buffers [1]. With our algorithm it is unnecessary to run long and expensive simulations for finite capacity queueing models.

ii) Extending the well-known "FLOW-EQUIVALENCY (also known Norton's Theorem) concept on finite capacity queueing models [2]. The major advantage of this technique is that computational expenses are reduced if only one or few stations from the queueing model are to be investigated under various system workloads.

iii) We also pointed out that deadlocks can occur in finite capacity queueing models with blocking [3]. We formally proved the deadlock freedom property and gave necessary and sufficient conditions for deadlock freedom.

iv) We demonstrated the application of our basic research results on several case studies [4,5,6].

Details about our contributions can be found in the attached papers [1-5].

3. APPLICATION EXAMPLES

In the following we discuss various examples from different areas such as airfleet, computer systems, communication networks, flexible manufacturing systems where our results can be applied.

3.1. Air Fleet Availability Analysis

Consider an example where we model a single flying base where operationally ready aircraft are stationed for training purposes and a repair depot where the aircraft are overhauled. The repair depot is represented as a series of stages and each stage represents the repair and replacement of a particular spare, and a repair shop and circulating spares are associated with each facility.

For example, a facility $i$ might represent engine shop, facility $ia$ might represent the shop for the engine and facility $j$, $ja$ for radio and so on. As an aircraft leaves the flying base it always needs an engine replacement and with a probability of $P_{ij}$ it needs a radio with $P_{ija}$ it needs a fuel pump with $P_{iua}$ it needs a gun sight and with probability $P_{ii}$ it needs a generator. After the repair at facility 6 it will go back to the flying base for
flight operations. Blocking can occur in this model, if for example, a spare is not available. In that case, the aircraft must wait in the according facility until the spare will be delivered.

Performance measures of the flying base, repair depot system are its productivity (total flight hours) and its operational effectiveness (availability). Availability is defined as the average fraction of aircraft available for use at a given instant. Several performance measures (e.g., average time an aircraft spends in the flying base and in the repair depot, average number of aircraft on ground at different stages of repair, use of repair facilities and average time an aircraft spends in various stages of repair) can easily be computed in the framework of our results. We can also determine the unavailability (out of stock probability) of a spare and the duration of its unavailability.

3.2. Interconnected Networks

An interconnected packet switching network consists of several local area networks which are connected by a long haul network. The local area networks are connected to a long haul network by gateways.

In the network there are two different types of communication:

i) Communication between hosts of each local area network (intranetwork traffic)

ii) Communication between hosts at different local area networks (internetwork traffic)

Hosts in the same local area network communicate with each other using a shared broadcast channel. The channel is accessed by the hosts via an interface, so-called network access unit (NAU). Based on communication protocols only one packet is allowed to be sent on the channel at a time. If a host wants to transmit a packet to another host in the same local area network it forwards it to its NAU. The access protocol of the local area network decides which packet will be transmitted next on the channel. All packets in NAU of the hosts can be seen as waiting in a global queue for accessing the channel. Once a packet obtains access to the channel it is immediately transmitted to the destination host, if source and destination hosts belong to the same local area network. If they do not belong to the same local area network, the packet is put into the NAU of the source local area network. The channel sends the packet to the gateway of the source local area network. The gateway then transmits the packet to the gateway of the destination local area network which forwards the packet to the according host through its broadcast channel.
In the queueing model the blocking occurs if a packet in any station is not allowed to leave if the destination station is full, i.e., the number of packets in the destination station is equal to its buffer capacity. In this case, the packet is blocked in the current station until a packet in the destination station is transmitted and a buffer space becomes available. Performance measures such as throughput, response time etc. can also be computed using our algorithms developed in this project. For example, we [4] applied our algorithm on this model and investigated different gateway topologies and determined their effect on the network performance.

### 3.3. Window Flow Controlled Packet Switching Networks

A queueing network model for a packet switching network with several virtual channels. Each virtual channel has a source and a sink. Packets in the same virtual channel follow a fixed route which may be chosen probabilistically from a finite set of routes between source and sink.

The delay for the return of an end-to-end (ETE) acknowledgement (ACK) from the sink to the source indicating receipt of a packet is modeled by an independent random variable, the distribution of which may be different for different virtual channels. This delay is modeled by an IS (infinite server) node that joins the sink to the source to yield a closed chain in the queueing network model. The flow control window size of a virtual channel is the maximum number of packets that it can have in transit within the communication network at the same time. If the number of packets in transit within a virtual channel is equal to its window size, then the source server is "blocked". A blocked source server is later unblocked when an ETE ACK returns from the sink indicating the receipt of a packet.

In [5] we demonstrated the applicability of our results on this type of model.

### 3.4. A Multiprocessor System Model

Consider a multiprocessor system consisting of several processors and several memory modules connected together by a multiplexed single bus. The memory modules have buffers at their inputs to queue the service requests of processors and buffers at their outputs to queue the requests served by the memory modules that cannot be served by the bus immediately. Assume a processor makes a request to a particular memory module. If the bus at that moment is not busy transferring a request for another processor or data from a memory module,
that processor takes the bus and the request is sent to that particular memory module. However, if the bus is busy transferring data, then that processor has to retry its request at a later time. If the memory module is free it will serve the request, if it is not free then the request will be queued. After the memory module completes its service, the output is placed in its output buffer for the bus, to be transmitted to the processor that made the request. The effect of a full node on its upstream nodes (nodes that have a directed arc to the full node) depends on the type of system being modeled. If the input buffers of the memory modules are full then the bus cannot place the request to the buffer, and the processor has to send a new request. The request will be transmitted a number of times until it is delivered by the bus when there is a space in the buffer. Similarly, the output buffer of a memory module can be full. In this case, the module may be forced to suspend its service until a request is delivered from its output buffer to the processor that made the request, i.e., until a space becomes available at the output buffer.

3.5. Database System Model

Users of a transaction processing system send their requests (read or update) to a request handler. When the request handler receives a request from a user, it evaluates the request and passes it to an appropriate server that is designed to handle that request type. Figure 6, being very simplistic illustrates two types of servers: one for handling inquiries and one for handling updates. The servers usually interact with the database manager to gain access to (or update) data in the database. It then formulates a reply and returns it to the user that made the requests via a reply handler.

In the queueing model of this system, the user requests are transmitted from user terminals to a request handler (REQ) over a communication channel (CC). Once the request is received and processed by the request handler, it is passed to the queue of the appropriate server: inquiry server (INQ) or update server (UP). The server processes the request and sends it to the data base manager (DBM) which accesses the disk service the request. Once the request is serviced by the disk, the data (for inquiry operation) or status (for update operation) is returned to the appropriate server. When the server completes its processing, it sends a reply to the reply handler (REP) which in turn returns the reply to the user via the communication link. The queues between various servers represent the buffers available for intermediate storage. Since there are a finite number of buffers available at each node, it is possible that one or more of the queues are full at any given time. In particular, a
user generates a request and attempts to access the communication channel. If the channel is busy transferring
another request and if there is a buffer available then the request is queued. However, if there is no buffer avail-
able then the user suspends its operation, i.e., it cannot generate new requests. Similarly, the communication to
the requests handler may be temporarily stopped if there is no space available in the request handler queue.
Furthermore, the communication channel cannot be used to store a request due to physical constraints, hence all
requests should wait in the queue until there is a space available in the requests handler’s queue at which time
the communication may be resumed.

REMARK. We have described several case studies where our basic research results can be applied. These case
studies are representative examples. Our results can be applied to other models as well.

4. REFERENCES

1. I. F. Akyildiz, Product Form Approximations for Queueing Networks with Multiple Servers and Blocking,
2. I. F. Akyildiz and J. Liebeherr, Application of Norton's Theorem on Queueing Networks with Finite Capaciti-
3. S. Kundu and I. F. Akyildiz, Deadlock Free Buffer Allocation in Closed Queueing Networks, *Queueing Sys-
   Report, Georgia Institute of Technology, School of Information and Computer Science, GIT-89-027, August
   1989.*
   *To Appear in INFOCOM’90 Computer Networks Conference, June 1990 in San Francisco.*
5. I. F. Akyildiz, Performance Analysis of Computer Communication Networks with Local and Global Window
Our investigation within this project was focused on two major areas: transfer blocking and rejection blocking protocols in queueing networks with finite capacities.

1. Networks with Transfer Blocking

In this type of networks blocking occurs when a job after completing its service at a station cannot proceed to its destination which is full. The job resides in the server of the source station and waits there until a place becomes available in the destination station.

1.1. Deadlocks

On page 5, equation (2), of the proposal [1] we stated that deadlocks may occur in this type of networks. We proved this statement formally in [2]. In [2] we also gave an algorithm for the distribution of total buffer capacities to stations such that no deadlock will occur. Additionally, we introduce an algorithm which automatically finds cycles in so-called cacti networks. This paper was submitted to "QUESTA: Queueing Systems: Theory and Applications" in October 1987 for publication. The paper is accepted in June 1988 and will be published soon.
1.2. Throughput Optimization

As we already observed in the investigation of queueing networks with blocking that the throughput is a non-decreasing function of the number of jobs [3], i.e., the blocking events have the effect of violating the throughput results which were mentioned on pages 7, 8 and 9 of the proposal [1]. Two questions arose from this observation:

i) How to distribute the total buffer capacity to the stations such that no deadlock will occur and a maximum (optimal) throughput will be achieved?

ii) Given the buffer capacity of each station in the network. How to select the total number of jobs in the network such that the throughput will be maximum (optimal)?

To answer these questions first we assumed that all stations have infinite capacity and derived new formulas for optimal throughput and response times based on the well-known mean value analysis approach [4]. Then in [5] we found necessary and sufficient conditions for buffer allocation in cyclic networks with blocking such that an optimal throughput will be achieved.

1.3. Norton's Theorem Application

Our major breakthrough for this proposal [1] was that we found a very efficient solution for the proposed problems stated on pages 5 through 11. The entire solution is described in detail in [6]. The execution of the application of Norton's theorem on blocking queueing networks as stated on pages 6 and 7 of [1], needs 4 steps for the analysis, (see page 7 of [6]):

i) Construction of the subnetwork \( \Gamma \)

ii) Throughput Analysis of the Subnetwork

iii) Construction of the Phases for the Selected Station

iv) Analysis of the Two-Station Network

As mentioned in the previous report for (AFOSR – 87–0160), the throughput algorithm suggested in the proposal [1] (on pages 7, 8, 9) and used in step ii) above, has extensively been studied for its accuracy. The algorithm is described in detail in [3].

Instead of determining the capacity of the composite station as proposed on page 9 in [1], we solved
that problem in a different way. We replace the subnetwork by a composite station with infinite capacity. However, the blocking events at selected station are neglected due to some full stations in the subnetwork. Therefore, we modify the service mechanism of the selected station such that all delays a job might undergo due to blocking events could be represented. By this modification we obtain a multiphase server representing all blocking delays in the selected station. We introduce an iterative technique to determine the parameters such as the branching probabilities and delay service times for the multiphase server of the selected station. Finally, the entire network was reduced to two-station network as it was planned in the proposal [1]. The reduced two-station network containing the selected finite capacity station with multiphase-server representing the blocking delays and the composite (flow-equivalent) infinite capacity station representing all other stations in the originally given network is analyzed by a numerical technique as discussed in [6].

The results of our technique have been validated by executing several examples and comparing them with simulation counterparts which are obtained by IBM-RESQ package. The validation study shows very good accuracy of our results.

2. Extended Parametric Analysis of Queueing Networks with Blocking (PART II)

In the first step we, in collaboration with colleagues from the University of Erlangen-Nuernberg, applied the extended parametric analysis concept on queueing networks with infinite capacities and implemented a parallel processing on a multiprocessor system with pyramid type architecture [7]. The experiments showed that parallel computing provides some advantages regarding storage space. The synchronization between processors take some time which reflects in run time of the programs. An open research question is how to optimize the run time as well as how to optimally decompose the network for parallel computation.

As proposed in section 4 in [1], we also applied the extended parametric analysis concept on queueing networks with blocking where the blocking network can arbitrarily be partitioned into disjoint subnetworks. A modified and extended version of the algorithm presented in [6] can be applied solving the problem of parallel analysis of the subnetworks. We briefly outline the major steps of the method.
using the example from section 4 in [1]. The network to be analyzed is given in Figure 1.

We decompose the network into four subnetworks $SNW_j$:

$SNW_1 = \{1, 2\}; \quad SNW_2 = \{12\}; \quad SNW_3 = \{3, 4, 5, 6, 7, 8, 9\}; \quad SNW_4 = \{10, 11\}$

Each subnetwork is analyzed by shortening all stations in other subnetworks, i.e., the service times of the stations not belonging to a subnetwork are set equal to zero. The analysis of the subnetworks is done as described in [6] and load-dependent throughput values $\lambda_j(k)$ are obtained for each subnetwork $SNW_j$ (for $j = 1, 2, 3, 4$). As pointed out in the proposal the analysis of each subnetwork is independent from other subnetworks. Therefore, this part of the algorithm can be carried out in parallel and speeding up the total computation.

In the next step we construct a composite station for each subnetwork $SNW_j$ with load-dependent service rates $\mu_{e_j}(k)$ which are obtained by setting $\mu_{e_j}(k)$ equal to the throughput values $\lambda_j(k)$. The composite stations are composed to a closed queueing network according to the configuration of the subnetworks in the originally given network. For the example given in Figure 1 we obtain a serially switched network as given in Figure 2.
The blocking events which occur during the transition of jobs traversing from one subnetwork to another are considered by adding delay phases to the load-dependent service stations in Figure 2. The construction of the delay phases is done as described in [6]. The application of our method to this example produces a network given in Figure 3. Note that the phase construction in Figure 3 is simplified in order to make a graphical representation possible.

The final step of the algorithm consists of an iteration between the network of Figure 3 and the analysis of each station of the originally given network in isolation (as a simple $M/M/1$ station with finite capacity). Trying to apply a similar method as described in [6] to this type of network we had to solve the following problems:

The multi-phase servers of the stations in Figure 3 contain load-dependent phases. We did not encounter this situation in [6] since the phase construction was only done for the selected station and not for a composite station representing a subnetwork. This situation is solved as follows. As in [6] for
the selected station we make the multi-phase servers computationally tractable by reducing the service and delay phases of each station to a Coxian server with two phases. The load-dependence is considered by a computation of the Coxian server under all possible loads. For the given example the following calculations are necessary:

\[ \mu_j(k) = 2 \cdot \tilde{\mu}_j(k) \]

\[ \tilde{\mu}_j(k) = \tilde{\bar{\epsilon}}^2_j(k) \cdot \mu_j(k) \]

\[ \tilde{a}_{j1}(k) = \frac{1}{2 \cdot \tilde{\bar{\epsilon}}^2_j(k)} \]

for \( j = 1, 2, 3, 4 ; k = 1, 2, \ldots, K \)

where \( K \) is the total number of jobs in the network. The values for \( \mu_j(k) \) and \( \tilde{\mu}_j(k) \) are obtained by calculating mean value and coefficient of variation of the multi-phase servers. This is straightforward since the phases are exponentially distributed and independent from each other. As a result we obtain a load-dependent server with a Coxian-2 service time distribution for each composite station. From [8] we know that this representation of a general service time distribution provides good results as long as the squared coefficient of variation for each \( \tilde{\bar{\epsilon}}^2_j(k) \) are greater than 0.5. In [6] only one subnetwork was constructed for the analysis. That one subnetwork was then analyzed by the numerical method. In this case, however, the numerical method is not applicable because of the dimension of the state space. By applying the load-dependent method of Marie [8] we obtained very accurate results for performance measures.

The applicability of the extended parametric analysis has numerous advantages. We mentioned the acceleration of computing time by carrying out subnetwork computations in parallel. Furthermore, some existing algorithms for very large finite capacity queueing networks with have overflows during the computation. Using the extended parametric analysis large networks can be decomposed into subnetworks and therefore they can become tractable for the analysis.

An application of the extended parametric analysis is given in [9]. There, we investigate the performance of internetwork traffic in a communication network where Local Area Networks (LANs) are connected using a Long Haul Network. We abstract from the individual structure of a LAN consisting
of a possibly large number of nodes by defining each LAN as a subnetwork which is applying the described method reducable to one composite station. Therefore, we are able to parametrize each LAN, thus allowing us to focus our study on the internetwork traffic without neglecting intranetwork traffic in the LANs.

3. Rejection Blocking

Within this project we also attacked queueing networks with rejection blocking. In this case blocking occurs when a job after completing its service finds its destination full. The job is rejected and goes immediately back to the server of the source station and receives another round of service. This is repeated until a space becomes available in the destination station. This type of blocking protocol appears in communication networks. For example, in a token ring network suppose a source node has the token and wants to send a message to a destination node. If the destination node does not receive the message, the value of the token remains unchanged and the token comes back to the source node which then assumes that its message has not been received. The source node tries so many times until the message will be received. This situation can be modeled by a queueing network with rejection blocking.

In [10] we give an algorithm for computation of performance measures in reversible networks. We also analyze non-reversible queueing networks in [10] where the analysis is based on the duality concept. Based on this concept we give exact solutions for performance measures. *This paper [10] has been presented at the International Workshop "Queueing Networks with Blocking" in May 1988 and it is appeared in the Proceedings of that conference.*

In [11] we studied queueing networks with mixed exponential and non-exponential parallel queues with interdependent service capacities and finite common pools and/or accessibility constraints. We showed that this type of networks have exact solution.

Note that our basic concept of parametric analysis as well as extended parametric analysis can easily be applied with some minor modifications also on networks of queues with rejection blocking.
References


4. I. F. Akyildiz and W. Liu, "Selecting Buffer Capacities in Cyclic Queueing Networks with Blocking", Short Note; Submitted for Publication.


CONFERENCES VISITED (Supported by AFOSR)


Presented the paper "General Closed Queueing Networks with Blocking"

Acceptance rate was 20%.


Presented the paper "Performance Analysis of Computer Communication Networks with Local and Global Window Flow Control",

Acceptance Rate was 60%.


Presented the paper "Analysis of Queueing Networks with Rejection Blocking",

Acceptance Rate was 50%.


Presented

i) "On Optimal Performance Measures of Computer Systems"

ii) "Open, Closed and Mixed Queueing Networks with Rejection Blocking"

Both papers were invited.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposal Cover Page</td>
<td>i</td>
</tr>
<tr>
<td>Extended Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Model Assumptions</td>
<td>3</td>
</tr>
<tr>
<td>3. Norton's Theorem Application on Blocking Queueing Networks (PROPOSED CONCEPT I)</td>
<td>5</td>
</tr>
<tr>
<td>3.1 Throughput Analysis of Blocking Networks</td>
<td>7</td>
</tr>
<tr>
<td>3.1.1 Determine the Number of States in Blocking Queueing Networks</td>
<td>7</td>
</tr>
<tr>
<td>3.1.2 Determine the Total Number of Jobs $R$ in the Equivalent Nonblocking Network</td>
<td>8</td>
</tr>
<tr>
<td>3.1.3 Determine the Throughput of the Equivalent Nonblocking Network</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Determining the Capacity of the Composite Station</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Analysis of the Two-Station Load Dependent Network</td>
<td>10</td>
</tr>
<tr>
<td>4. Extended Parametric Analysis of Blocking Queueing Networks (PROPOSED CONCEPT II)</td>
<td>11</td>
</tr>
<tr>
<td>4.1 Computation of the Load Dependent Throughputs for the Subnetworks</td>
<td>14</td>
</tr>
<tr>
<td>4.2 Composing of Subnetworks</td>
<td>14</td>
</tr>
<tr>
<td>4.3 Analysis of Serial Order</td>
<td>15</td>
</tr>
<tr>
<td>5. Validation</td>
<td>15</td>
</tr>
<tr>
<td>References</td>
<td>15</td>
</tr>
<tr>
<td>Resume</td>
<td>18</td>
</tr>
<tr>
<td>Budget</td>
<td>22</td>
</tr>
</tbody>
</table>
1. Introduction

In recent years there has been a growing interest in the development of computational methods to analyze queueing networks with blocking. These are networks where the stations have finite capacities, hence blocking can occur if the station is full to capacity. A job which wants to come to the full station must reside in its source station and block the source station until a place is available in the destination station. The interest in networks with blocking comes primarily from the realization that these models are useful in the study of the behavior of subsystems of computers and communication networks, in addition to detailed descriptions of several computer-related applications such as flexible manufacturing systems.

Blocking networks are classified into 3 different types [ONVU86].

a) Transfer Blocking. In this case, the blocking event occurs when a job completing service at station $i$ cannot proceed to station $j$ because station $j$ is full. The job resides in station $i$'s server, which stops processing until station $j$ releases a job. This type of blocking has been used to model systems such as production systems and disk I/O subsystems. [AKYL87a,b,c,d,e, ALTI86, BRAN85, ONVU87, PERR81, PERR86, PERR87, SURI86, TAKA80].

b) Service Blocking. In this case, blocking occurs when a job in front of queue at station $i$ declares its destination station $j$ before it starts its service in station $i$'s server. If the destination station $j$ is full, the $i$-th server becomes blocked, i.e., it cannot serve jobs. When a departure occurs from destination station $j$, the $i$-th server becomes unblocked and the job begins receiving service. This blocking type has been used to model systems
such as telecommunication and production systems, [BOXM81, GORD67, KONH76, KONH77, SURI84].

\textit{c) "Rejection Blocking".} In this case, blocking occurs when a job completes service at station \(i\)'s server and wants to join station \(j\), whose capacity is full. The job is rejected by station \(j\). That job goes back to station \(i\)'s server and receives a new service with the same mean service time. This activity is repeated until station \(j\) releases a job, and a place becomes available. The "rejection blocking" type has been used to model systems such as communication networks, computer systems with limited multiprogramming, production lines and flexible manufacturing systems. [AKVO87a, AKVO87b, BALS83, HORD81, PITT79, VONB87].

Formal comparisons between these distinct classes of blocking have been carried out extensively by Onvural/Perros [ONVU86]. Several investigators in recent years have published results on queueing networks with transfer, service and rejection blocking. A bibliography concerning queueing network models with blocking is given by Perros [PERR84]. A survey concerning the blocking networks is given by Perros [PERR85], Onvural [ONVU87] and Akyildiz/von Brand [AKVO87c].

Since we are investigating the networks with transfer blocking we discuss here the previous work for this type of networks.

In [AKYL87a] we studied two-station closed queueing networks with classical blocking and multiple server stations. We have shown that the equilibrium state probability distributions of such blocking systems are identical to those of a two-station closed queueing network without blocking. In [AKYL87b] we show that the throughput of a blocking network with \(K\) total number of jobs is approximately equal to the throughput of a nonblocking network with an appropriate total number of jobs \(K'\), which can be easily calculated. In [AKYL87d] we extend the well-known mean value analysis algorithm [REIS80] to single server queueing networks with blocking. The approximation is based on the modification of mean residence times due to the blocking events that occur in the network. In [AKYL87e] we derive a new algorithm for the computation of throughput values and the mean queue lengths in blocking networks with general service time distributions and FCFS scheduling disciplines.

Takahashi, Miyahara and Hasegawa [TAKA80] developed a method for approximate analysis of open queueing networks with classical blocking. Each station is treated as an M/M/1 finite capacity queueing system whose arrival rate and mean service time are expressed in terms of the blocking probabilities. These probabilities are in turn expressed in terms of the arrival rates and mean service times yielding a set of \(N\) simultaneous non-linear
equations whose solution yields an approximation for blocking probabilities. Approximations for other performance measures can be obtained from these probabilities. However, only a very limited accuracy assessment was performed.

Perros [PERR81] considered a general class of open exponential queue networks consisting of \( n \) (\( n \geq 2 \)) queues in parallel being serviced by servers who form a hierarchical structure. Blocking of a server occurs each time the server completes a service. The server remains blocked until its blocking unit departs from the network having received service by all the other servers to which this server is linked. Approximate and exact results for the utilization of a station were obtained. Perros/Altio [PERR86] analyze open queueing networks with exponentially distributed service times. Their work is based on communication networks in which each station is serially switched. A Cox distribution for each station is developed in which the second phase represents the blocking phase of the corresponding station. Blocking probabilities are determined using an iterative formula.

Several other investigators in recent years have published results on queueing networks with rejection as well as classical blocking. An excellent bibliography concerning queueing network models with blocking is given by Perros [PERR84]. Our literature review suggests that existing or proposed methods either contain disadvantages (e.g., long run times and/or memory space) and/or restrictions (only two-station or tandem network solutions) or provide approximate results which differ widely from the exact values. The blocking network models which have been investigated so far have the following additional limitations:

i) All service time distributions are exponential.

ii) The queueing discipline at each station is basically FCFS.

iii) All stations may have single (load independent) servers.

In this proposal we will attack these limitations and propose new solutions for closed queueing networks with blocking.

2. Model Assumptions

We consider closed queueing networks with \( N \) stations and \( K \) total jobs. Each station in the network may have the following four station types:
- 4 -

• Type 1a. \[ M / M / 1 - FCFS \]

• Type 1b. \[ M / M / id - FCFS \] (id: load-dependent server; allows multiple servers)

• Type 2. \[ \psi / G / 1 - PS \]

• Type 3. \[ \psi / G / IS - .. \]

• Type 4. \[ \psi / G / LCFS - PR \]

Each station of Type 1a or Type 1b has exponential service time distribution and of Type 2,3,4 general service time distribution with mean values \( 1/\mu_i \) for \( i = 1, \ldots, N \). The service rate of Type 1b station is load dependent \( \mu_i(\xi) \). Note that four types of stations are motivated by some practical considerations. Type 1 stations are useful in many instances (secondary memory units, input-output devices, etc.). Type 2 stations are, in many cases, a reasonable representation of the CPU allocated in quanta; the processor sharing discipline is an idealization with a quantum of "infinitesimally small" (in fact, zero) duration and no overhead associated with switching from one job to the other. A Type 3 station represents well terminals in a time-sharing system. Type 4 stations can be used to represent stacks in data structure models.

Each station also has a fixed finite capacity \( M_i \) where \( M_i = (\text{queue capacity} + 1) \), (for \( i = 1,2,\ldots, N \)). Cases in which the stations can have infinite capacity are also allowed. \( M_i = \infty \), (for some \( i = 1,2,\ldots, N \)). Any station whose capacity exceeds the total number of jobs in the network can be considered to have infinite capacity. A job which is serviced by the \( i \)-th station proceeds to the \( j \)-th station with probability \( p_{ij} \), (for \( i, j = 1,2,\ldots, N \)), if the \( j \)-th station is not full. That is, if the number of jobs in the \( j \)-th station, \( k_j \), is less or equal to \( M_j \) for \( j=1,2,\ldots, N \). Otherwise, the job is blocked in the \( i \)-th station until a job in the \( j \)-th station has completed its servicing and a place becomes available.

Furthermore, we assume that

\[
K < \sum_{i=1}^{N} M_i
\]

which implies that the total number of jobs \( K \) in the network may not exceed the total station capacity of the entire network.

One of the most important problems to realize regarding blocking queueing networks is that finite station capacities and blocking can introduce the problem of system deadlock. Deadlock may occur if a job which has
finished its service at station $i$'s server wants join station $j$, whose capacity is full. That job is blocked in station $i$. Another job which has finished its service at $j$-th station now wants to proceed to the $i$-th station, whose capacity is also full. It blocks station $j$. Both jobs are waiting for each other. As a result a deadlock situation arises. In [AKYL86a] we have demonstrated the necessary conditions for a closed queueing network with single job class to be deadlock free. The following assumption states that a closed queueing network containing finite station capacities is deadlock free if and only if for each cycle $C$ in the network the following condition holds:

$$K < \sum_{j \in C} M_j$$

Simply stated, the total number jobs in the network must be smaller than the sum of station capacities in each cycle.

Equation (1) is a sufficient condition for tandem networks to be deadlock free.

With these assumptions we obtain the queueing network model, classified as classical blocking which will be the object of our investigation.

3. Norton's Theorem Application on Blocking Queueing Networks (PROPOSED CONCEPT I)

The parametric analysis is based on an application of Norton's Theorem from electrical circuit theory to queueing networks. Chandy, Herzog and Woo [CHAN75] showed that Norton's Theorem provides an exact analysis of queueing networks, if such networks have a product form solution. We explain this concept by considering a closed queueing network model with $K$ jobs and $N = 12$ stations shown in Figure 1.

![Figure 1.](image-url)

With this queueing network model we can construct an equivalent network in which we arbitrarily select a
station \( N \), and replace the other \((N - 1)\) stations by a single station, called the composite (flow-equivalent) station as shown in Figure 2.

Let \( \mu_c(k) \) be the composite mean service rate, where \( k \) is the number of jobs at this composite station. These composite mean service rates \( \mu_c(k) \), for \( k = 1, \ldots, K \), are determined by analyzing a modified version of the given network, in which the selected station \( N \) has been shorted as shown in Figure 3.

The mean service time of that station \( 1/\mu_r \) is set equal to zero and the throughputs \( \lambda(k) \), of the shorted system for all jobs \( k = 1, 2, \ldots, K \) are calculated. These computed throughputs through the shorted network \( \lambda(k) \), are set equal to the composite mean service rates \( \mu_c(k) \). The solutions of the network consisting of the selected and composite stations are identical to those of the originally given network model.

The parametric analysis of Chandy, Herzog and Woo [CHAN75] is very interesting for cases in which only one station (e.g., a CPU) in a queueing network model is to be analyzed under various system workload.

The parametric analysis of blocking queueing networks is executed as follows:

3.1. Computation of the Throughput Values of the Subsystem
3.2. Determining of the Composite Station Capacity

3.3. Analysis of the Two-Station Load Dependent Network

3.1. Throughput Analysis of Blocking Networks

We already have an algorithm to calculate the throughput of a blocking network with single server stations [AKYL87b]. Here we propose an extension of that algorithm for blocking networks with multiple servers.

The basic concept is that the state space of the blocking queueing network with $K$ total number of jobs is transformed into the state space of a nonblocking queueing network with $K^2$ total number of jobs. The number of states in both networks should be approximately the same, if not identical. This would indicate that Markov processes describing the evolution of both networks over time have an almost identical structure. That, in turn, would guarantee that the throughputs of both systems are almost equal.

The following steps are executed in order to compute the throughput values in queueing networks with blocking.

3.1.1. Determine the number of states in blocking queueing network.

3.1.2. Determine the total number of jobs $K^2$ in the equivalent nonblocking queueing network.

3.1.3. Analyze the nonblocking queueing network with $K^2$ jobs to obtain the throughput values which are equal to the throughput value of the blocking network with $K$ jobs.

3.1.1. Determine the number of states in blocking queueing networks

As previously mentioned, in blocking networks each station has a capacity limit, which indicates that only a certain number of states can be feasible. The feasible states for blocking networks are obtained by realizing that the number of jobs in the $i$-th station $k_i$ may not exceed its capacity $M_i$, $k_i \leq M_i$.

Blocking events which occur in networks with finite station capacities must also be taken into account. Therefore, the $m_i$ (number of servers) neighbors of the feasible states are included, representing the blocking states. Whenever a transition occurs from one state to another state in which the capacity limit of a station would be violated, we assume that the transition causes a blocking action in the network and that the state entered is a
blocking state. In reality, the job still resides in the source station. All the other states are infeasible and are cancelled.

Using this method we obtain a sub-state space for the blocking network. From the reduced state space we can obtain the number of states $Z'$ of the blocking network, which is the sum of feasible states and blocking states. Since the number of states $Z'$ can be very large for general networks we cannot draw the state space, eliminate the nonfeasible states or count the total number of feasible states and their neighbors as blocking states in an efficient way. In order to directly obtain the number of states $Z'$ in queueing networks with blocking, we have developed an efficient convolution algorithm [AKYL87b] which is applicable only to networks containing Type 1a stations. We must find an efficient algorithm which provides the number of states $Z'$ (the feasible states and their neighbors as blocking states) in closed queueing networks with four different finite capacity station types.

### 3.1.2. Determine the total number of jobs $k'$ in the equivalent nonblocking network

Our primary objective is to find an equivalent nonblocking network which has the same number of states and the same state space structure as the blocking network. In general, the state space of the blocking network cannot be transformed bijectively into the state space of an equivalent nonblocking network. However, the number of states in both systems may be equal or almost equal. This would imply that both systems have the same behavior and the throughputs of both systems are almost identical. Assume that the number of states $Z'$ (feasible and blocking states) in the blocking queueing network is obtained somehow. The goal is to find a total number of jobs $k'$ in an equivalent network with infinite station capacities which will provide almost the same number of states, $Z$, as in the blocking network. We then find an appropriate total number of jobs $k'$ in the equivalent nonblocking network such that $Z$ will be approximately equal to $Z'$.

Since the number of states in both systems will be equal or almost equal, it implies that the Markov processes describing the evolution of both networks have approximately the same behavior. Consequently, the throughput of the equivalent nonblocking network $\lambda_{NB}(k')$ is almost equivalent to the throughput of the blocking network $\lambda_{b}(K)$.

### 3.1.3. Determine the throughput of the equivalent nonblocking network

By analyzing the equivalent nonblocking network with $k'$ total jobs using a product form algorithm such as mean value analysis, [REIS80], we obtain the total throughput $\lambda_{NB}(k')$. This is almost identical to the total
throughput $\lambda_g(K)$ of the blocking network with $K$ total number of jobs [AKYL87b].

$$\lambda_g(K) = \lambda_{NB}(K)$$  \hfill (3)

Note that we do not need any other performance measures than the throughput of the subsystem. By varying the number of jobs in the subsystem, from 1 to $K$, we can obtain $\lambda_g(1), \lambda_g(2), \ldots, \lambda_g(K)$. Note that the throughput values $\lambda_g(k)$ for $k = 1, \ldots, K$ for the blocking network are approximate. Consequently the parametric analysis for blocking networks will provide approximate results.

3.2. Determining the Capacity of the Composite Station

The given system has been reduced to a two-station blocking network. The service rate of composite station $\mu_c(k)$ is load-dependent, and set equal to the computed throughputs $\lambda_g(k)$ for $k = 1, \ldots, K$. The major problem which arises is the capacity of the composite station. Initially, we define the capacity of the composite station as:

$$M_a = \sum_{j \in \sigma} M_j$$  \hfill (4)

where $\sigma$ represents the subsystem.

Our experience shows that this overestimates throughput, since the shorted station can be blocked in the actual network with less than $M_j$ jobs in the subsystem $\sigma$.

Another possibility is using a capacity weighted by the transition probabilities from the shorted station $i$ to the stations in the subsystem:

$$M_a = \sum_{j \in \sigma} M_j \cdot p_{ij}$$  \hfill (5)

However, we have discovered that this underestimates the throughput.

Note that Suri/Diehl [SUR986] realized also this fact and solved this problem by constructing so-called "views" as seen by the previous station. They assume that the shorted station $i$ must have infinite capacity. The "view" seen by station $i$ is for $k$ jobs in the successor stations sometimes sees the station $i$ as blocked and sometimes unblocked. Thus, the shorted station $i$ sees a finite buffer of variable size $k$ in the composite station. They introduce a so-called variable buffer size model which represents the view seen by the shorted station $i$. An iterative formula is used to compute the views. There are some limitations in their work. The model is restricted to single server tandem networks, in particular, the job flow can only be in one direction. The network cannot have arbitrarily
switched stations. Another limitation is that the shorted station must have an infinite capacity. Their work will definitely help us in our investigations.

3.3. Analysis of the Two-Station Load Dependent Network

In [AKYL87], we have shown that two-station closed queueing networks with blocking have an exact product form solution. The solution concept is based on the transformation of the state space of the blocking queueing network into a state space for a nonblocking network. The state space of two-station queueing networks is one-dimensional. It is easy to find an equivalent nonblocking network which has exactly the same structure as the blocking network. In [AKYL87], we have proved the following theorem:

Theorem. For a two-station closed queueing network with classical blocking there exists an equivalent two-station closed queueing network without blocking having the same structure. The equilibrium state probabilities \( p(k_1, k_2) \) for the blocking network are computed by the product form solution for the equivalent nonblocking network.

In the load-dependent case, the following problem can cause the results for two-station network to be only approximate. State transitions of the blocking and equivalent nonblocking network do not agree. This will be explained by a numerical example. Assume a two-station network with 6 jobs are given. The first station has the capacity \( M_1 = 4 \) and the second station \( M_2 = 3 \). The service rate of the first station is load-independent \( \mu_1 \) where the second station has load-dependent service rates \( \mu_2(k) \). The state space diagram for the above network is:

![State Space Diagram](image)

Figure 4.

By considering the station capacities the following sub-state space is obtained shown, in Figure 5, containing the feasible states and the blocking states (denoted by *) for the blocking network.
Now we find a nonblocking network with an appropriate number of jobs $k^2$ which provides the same state space structure as the blocking network. Since there are $Z' = 4$ states in Figure 5, we find that $k^2 = 3$. The state space for $k^2 = 3$ jobs is shown in Figure 6.

However, the transition rates between states in Figure 6 and 5 do not agree. This can cause the results to no longer be exact. We can attempt to overcome this problem by having the same transitions between the states of the nonblocking network as shown in Figure 7.

Figure 7 and Figure 5 have exactly the same structure and the same transition rates. On the other hand, since we have a network with load-dependent service rates, transition rates of Figure 6 are more realistic than those of Figure 7. We will investigate further on which diagram should be analyzed, and explore the possibility of extending our exact algorithm [AKYL87a] to handle the "load-dependent" case.

4. Extended Parametric Analysis of Blocking Queueing Networks (PROPOSED CONCEPT II)
Some papers extending Norton's theorem for queueing networks with infinite capacity have been published in the recent years. Kritzinger/van Wijk/Krzesinski [KRIT82] have extended the work of [CHAN75] to closed, open and mixed queueing networks with multiple job classes. Balsamo/Izeolla [BALS82] partition a network with $N$ stations and $K$ jobs into two subsystems where the first subsystem contains the stations whose behavior is to be studied and the second subsystem represents the uninteresting part, i.e., stations whose behavior is not of interest. Their method is based on the matrix of the transition probabilities. They eliminate one uninteresting station by setting its mean service time equal to zero, and obtain a new transition probability matrix for the jobs. They repeat this elimination procedure for the next uninteresting station and construct a new transition probability matrix for the jobs. This elimination procedure must be repeated and a new transition probability matrix must be constructed for each station as they want to eliminate. This method is complex and requires a large amount of computation time for large queueing networks.

We will extend the parametric analysis of blocking queueing networks as proposed in section 3, in which a queueing network model can arbitrarily be partitioned into $S$ disjoint subnetworks, in short form $SNW_j$ (for $j = 1, 2, \ldots, S$), each containing multiple stations. This concept is illustrated by the example given in Figure 1. Figure 8 shows the given queueing network from Figure 1 decomposed into 4 subnetworks.

Each subnetwork $SNW_j$ is analyzed by shorting all other stations in all other subnetworks, i.e., their service times are set to zero. Since the subnetworks can be analyzed independently, this analysis can be executed in parallel. As a result, the load dependent throughput values $\lambda(k)$ for each subnetwork are obtained simultaneously. Each subnetwork $SNW_j$ containing multiple stations can thus be composed into a single station. The com-
puted load dependent throughputs $\lambda(k)$ for each $SNW_{j}$ (for $1 < j < S$) are set equal to the load dependent service rates $\mu_{c}(k)$ of the respective composite station. The composite stations are serially switched and the simplified network, shown in Figure 10, is easily analyzed when computing all relevant characteristic performance measures. These measures are valid for each station of the originally given network model, Figure 1.

![Figure 9](image)

This extended parametric analysis is not only motivated only by the desire to accelerate the processing speed and reduce the memory space, but also by the fact that some stations (more than one) could be studied under various system input parameters in which the remaining subsystem is represented by a composite station containing all stations whose behavior does not interest us.

As a formal example, assume that stations 1 and 2 in Figure 1 represent two independent CPU’s. If we wish to investigate only these CPU’s under various workload parameters, we do not need to consider the remaining stations (3 through 12) in our computations. It is sufficient to initially analyze the remaining system computing the composite load dependent throughput values $\lambda(k)$ by setting the mean service times of both CPU’s (stations 1 and 2) to zero. In this way we obtain Figure 10, in which the composite station represents stations 3 through 12.

![Figure 10](image)

The mean service times of the CPU’s (stations 1 and 2) are stated initially, while the mean service rate of the composite station is equal to the computed load dependent throughput value $\lambda(k)$. This queueing network model, Figure 10, can be used for the analysis of the stations 1 and 2 under various system input parameters.

The extended parametric analysis is realized in three steps:
4.1. Computation of the load dependent throughputs

In order to analyze a subnetwork $SNW_j$, we must short all stations which do not belong to that subnetwork $SNW_j$. We then compute the load dependent throughputs for each subnetwork $SNW_j$ for $(1 < j < S)$. For the computation of this measure we will use the algorithm proposed in section 3.1. Note that each subnetwork can be analyzed independently from the others. The independent analysis of each subnetwork can be executed in parallel in order to accelerate the processing speed. In the infinite capacity network case we realized this parallel execution on 4 processors and reached an acceleration of the computations by a factor of 3 to 3.3 [AKYL86b]. An optimal decomposition aggregates the entire network into multiple subnetworks such that the following relation is valid:

$$\text{[Number of Subnetworks]} \mod n \geq (n - 1)$$

where $n$ is the number of processors.

The purpose of this relation is to prevent processors from waiting on the results of other processors.

4.2. Composing of Subnetworks

We compose the stations of each subnetwork into one station. The mean service rate $\mu_j(k)$ of this composite station is dependent on the number of jobs in the subnetwork and is given by the throughput of the composite subnetwork.

$$\mu_j(k) = \lambda_{SNW_j}(k) \quad \text{for } j = 1, \ldots, S.$$  

These composite stations are switched serially in an arbitrary order.

The same problem occurs here arises with the capacity of each composite station. The solution of section (3.2) will provide an answer to this problem.
4.3. Analysis of Serial Order

For the analysis of these arbitrarily ordered serially switched composite stations we apply the algorithm suggested in [AKYL87c]. Note that the algorithm suggested in [AKYL87c] is applicable to Type 1a stations. However, it can easily be extended such that other types of stations can also be analyzed.

REMARK

The Extended Parametric Analysis concept can be applied in order to simplify the computational requirements involved in large queueing network models with blocking. Using this concept, the large storage requirement and the long run times of the existing algorithms, in particular of our algorithm [AKYL87c] can be reduced drastically. This is due to the fact that the one large queueing network is analyzed as multiple small independent networks.

The major advantage of this technique is that computational expenses are reduced if only a few stations from the queueing network model are to be investigated under various system workloads. In this case we must determine the throughput values of the subnetwork which contains the interesting stations. These throughput results are then used for the analysis of the remaining subnetworks. The advantage results from the fact that the throughput values for the remaining subnetworks are computed only once in the beginning and remain fixed under various system workloads.

5. Validation

Our solutions in both cases (Parametric and Extended Parametric Analysis) will be approximate. For validation all suggested formulas and the proposed algorithms must be tested and implemented. A large variety numerical examples, including several stress tests, must be executed and then compared with simulation results. The RESQ [SAUE81] simulation package is used to simulate the blocking networks.

References


Deadlock Free Buffer Allocation in Closed Queueing Networks

I. F. Akyildiz* and S. Kundu**

* School of Information and Computer Science
  Georgia Institute of Technology
  Atlanta, Georgia 30332
  U. S. A.

** Department of Computer Science
  Louisiana State University
  Baton Rouge, Louisiana 70803
  U. S. A.

ABSTRACT

Blocking queueing networks are of much interest in performance analysis due to their realistic modeling capability. One important feature of such networks is that they may have deadlocks which can occur if the node capacities are not sufficiently large. A necessary and sufficient condition for the node capacities is presented such that the network is deadlock free. An algorithm is given for buffer allocation in blocking queueing networks such that no deadlocks will occur assuming that the network has the special structure called cacti-graph. Additional algorithm which takes linear time in the number of nodes, is presented to find cycles in cacti networks.

Key Words: Performance Evaluation, Queueing Networks, Finite Buffers, Blocking, Deadlock

August 11, 1988

Akyildiz's work was supported in part by School of Information and Computer Science, ICS, of Georgia Tech and by the Air Force Office of the Scientific Research (AFOSR) under Grant AFOSR-88-028.
Deadlock Free Buffer Allocation in Closed Queueing Networks

I. F. Akyildiz* and S. Kandu**

* School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

** Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70803
U. S. A.

1. Introduction

Since in actual systems the resources have a finite capacity, queueing networks with blocking must
be used for performance analysis. In queueing networks with blocking, a node can be thought of as a
device with a finite length queue. The network is simply a set of arbitrarily linked nodes. Blocking
arises due to the limitations imposed by the capacity of these nodes. In particular, blocking occurs when
the flow of jobs through one node is momentarily suspended due to the fact that another node has
reached its capacity limitation.

Several papers have been published dealing with various types of blocking. Previous work regard-
ing the blocking networks falls into three classes, Onvural and Perros [10]:

i) Transfer Blocking. Upon completion of the service at node \( i \), a job attempts to enter the
destination station \( j \). If node \( j \) is full at that moment, the job is forced to wait in node \( i \)’s
server until it can enter destination node \( j \). The server remains blocked for this period of
time. It cannot serve any other jobs waiting in the queue. This type of blocking has been
used to model systems such as production systems and disk I/O subsystems, Akyildiz [1,2],
Perros and Altiok [12].

ii) Service Blocking. A job at node \( i \) declares its destination node \( j \) before it starts its service. If
node \( j \) is full, the \( i \)-th server is blocked before service begins. When a departure occurs
from destination node $j$, the $i$-th server becomes unblocked and the job begins receiving service. This blocking type has been used to model systems such as production systems and telecommunication systems, Boxma and Konheim [5], Gordon and Newell [6], Konheim and Reiser [9].

iii) **Rejection Blocking.** Upon service completion at node $i$, a job attempts to join destination node $j$. If node $j$ is full at that moment, the job receives a new service at node $i$. This is repeated until the job completes service at a time when it can proceed to station $j$, Balsamo and Iaseolla [3], Hordijk and van Dijk [7].

Several other investigators in recent years have published results on blocking queueing networks, Perros [11].

An important consideration in blocking queueing networks of any type is that finite node capacities and blocking can introduce the deadlock situation. In a simple example, deadlock may occur if a job which has finished its service at node $i$'s server wants to join node $j$, whose capacity is full. That job is blocked in node $i$. Another job which has finished its service at $j$-th node now wants to proceed to the $i$-th node, whose capacity is also full. It blocks the $j$-th node. Both jobs are waiting for each other. As a result, a deadlock situation arises. Furthermore, the possibility of deadlock in a network increases with the ratio of the number of jobs in the network to the total capacity of the network. As the total number of jobs approaches the total capacity of the network, the chance of deadlock increases.

The most important issue is the allocation of node capacities in a queueing network such that deadlocks cannot occur. In this work we give a necessary and sufficient condition for a queueing network to be deadlock free, and present an algorithm for computing the capacities for the nodes such that no deadlock will occur in the network.

2. **Deadlock Freedom in Blocking Networks**

Let $\Gamma$ be a closed queueing network of Type 1 with $N$ nodes and $K$ jobs where all jobs are of the same class. Each node contains $m_i \geq 1$ servers with a single queue. There are no restric-
tions regarding the service time distributions and scheduling disciplines of the nodes. Let $B_i$ be the buffer size, or capacity, of the $i$-th node where $B_i = \text{Queue Capacity}_i + m_i$ (for $i = 1, ..., N$). There can be at most $B_i$ jobs at node $i$ at any time, including the jobs which are currently being serviced. A job which is serviced by the $i$-th node proceeds to the $j$-th node with probability $p_{ij}$ for $(i, j = 1, 2, ..., N)$, if the number of jobs at the $j$-th node has not exceeded the capacity $B_j$. Otherwise, the job is blocked at the $i$-th node until a job at node $j$ has completed service and a place becomes available. This model is classified as Type 1 blocking above. It is understood that once a job selects a destination (probabilistically or deterministically) it cannot change the destination. This is implication in Type 1 blocking network definition. We assume that each job has a fixed class assigned to it and this cannot change because it is blocked at some point in time.

The following theorem describes a necessary and sufficient condition for a closed queueing network to be deadlock free. A cycle $C$ is a sequence of nodes $(x_1, x_2, ..., x_M)$ such that each pair of consecutive nodes is joined by an arc $(x_i, x_{i+1})$, including the arc $(x_M, x_1)$.

**Theorem 1.** A closed queueing network of Type 1 with finite node capacities $\{B_i : 1 \leq i \leq N\}$ is deadlock free if and only if for each cycle $C$ in the network the following condition (1) holds. Simply stated, the total number of jobs in the network must be smaller than the sum of node capacities in each cycle.

$$K < \sum_{j \in C} B_j$$

(1)

**Proof.**

i) **Necessity.** Suppose that there is a cycle $C = (1, 2, \ldots, M)$, $(M \leq N)$, which violates the condition (1). Consider a state of the network in which each node $i$ in $C$ is saturated, i.e., the current number of jobs at node $i$, $1 \leq i \leq M$, equals its buffer capacity $B_i$. There is a positive probability for such a state of the network since $K \geq \sum_{j \in C} B_j$.

Now, assume that for each node $i$ in $C$, the job which is currently being serviced at $i$ finishes and it wants to move to the next node $i + 1$ in the cycle ($M + 1 = 1$). There is
also a positive probability for this to happen. This, however, results in a deadlock within the cycle C. Since there is also a positive probability that a job in another node may want to move to a node in C, eventually all nodes will be deadlocked with probability 1.

ii) Sufficiency. Suppose that there is a blocking. For example, the node 1 is blocked. Then there is another node 2 such that the job at node 1 which has completed service wants to move to node 2 cannot do so. This means that node 2 is saturated and must itself be blocked. Otherwise, at some point in the future, the current job at node 2 would move out, and the job from node 1 could then move to node 2. By repeating the above argument for node 2 and so on, we get a sequence of nodes (1, 2, \cdots) with the following properties:

a) Each node i is blocked and is saturated.

b) (i, i+1) is an arc of the network \( \Gamma \).

Since \( \Gamma \) is finite, the nodes \( \{i: i \geq 1\} \) must include a cycle, \( C = \{1, 2, \ldots, M\} \), without loss of generality. Since each \( i, 1 \leq i \leq M \), is saturated, we have

\[
K \geq \sum_{j \in C} B_j
\]

(2)

This violates the inequality (1) for the cycle C, a contradiction. This completes the proof.

If \( \Gamma \) is a tandem network, consisting of a single cycle, then the inequality (1) corresponds to the total buffer size \( B = \sum_{i=1}^{N} B_i \) of the network being at least \( (K + 1) \). This can be achieved by taking \( B_1 = B_2 = B_{(N-1)} = 1 \), and \( B_N = K - N + 2 \). (Indeed, a better throughput may be achieved by allocating buffer sizes at the nodes in inverse proportion to their service rates. This has been verified in some experimental cases, but has not been established formally. The issue of buffer allocation problem for improving throughput is not considered here.)

Corollary. A necessary and sufficient condition for a tandem network to be deadlock free is

\[
K < \sum_{i=1}^{N} B_i
\]

(3)
3. Deadlock Free Buffer Allocation in Blocking Networks

A set of buffer sizes \( \{B_i : 1 \leq i \leq N \} \) for which inequality (1) holds for every cycle \( C \) is called a deadlock free buffer allocation, or dfba. We denote by \( \beta \) the minimum value of \( B = \sum_{i=1}^{N} B_i \), taken over all dfba's for the network. The minimum buffering requirement of the network for avoiding deadlocks is \( \beta \). It is clear that for a buffer allocation \( \{B_i : 1 \leq i \leq N \} \) to be deadlock free, it needs to satisfy only those inequalities in (1) which correspond to the elementary cycles, i.e., the cycles which do not pass through the same node more than once. We assume here that each node and each arc of the network belongs to at least one cycle; a node, however, can belong to several cycles. We give below an algorithm for computing \( \beta \) and a corresponding buffer allocation \( \{B_i : 1 \leq i \leq N \} \) for the case where the network \( \Gamma \) has the form of a tree of elementary cycles. Figure 1 shows such a network, where the direction of each arc is clockwise around the cycle. Such a network is called a cactus network, Behzad et. al. [4]. A cactus has the property that no two cycles have more than one node in common. We define the following terms to describe the algorithm:

i) A contact node is a node at which two or more cycles meet. Since we assume that every node belongs to a cycle, this is the same as saying that there are 2 or more arcs leaving the node.

ii) A terminal cycle is a cycle which contains at most one contact node. Unless the cactus is a cycle, a terminal cycle has exactly one contact node, and there are at least two terminal cycles in a cactus. In Figure 1, the cycles (1, 2, 3, 4) and (7, 8) are two of the four terminal cycles, with the contact nodes 4 and 8, respectively.

The algorithm BUFFER given below assigns values to \( B_i \) in an "outside-in" fashion. The correctness of the algorithm is based on Lemma 1 which shows that given any terminal cycle \( C \) there is a deadlock free buffer allocation where all nodes other than its contact node (if any) has buffer size 1. The step (d) of the algorithm makes use of this principle. The other steps of the algorithm extends the current buffer allocation gradually until the condition (1) is satisfied for all
cycles.

Algorithm BUFFER(N):

(a) Assign $B_i = 1$ initially for each node of the network.
(b) Choose a terminal cycle $C$ of the network.
(c) If $C$ is equal to the entire network, and $C$ does not satisfy (1), then increase the buffer size at one of its nodes to make $\sum_{i \in C} B_i = (K + 1)$, and stop.
(d) Let $i$ be the contact node of $C$. If $C$ does not satisfy (1), then increase $B_i$ such that $\sum_{i \in C} B_i = (K + 1)$
(e) Delete all nodes in $C$ from the network, except the node $i$.
(f) Repeat steps (b)-(e) until "stop" is encountered.

If we are given an existing buffer allocation in which the condition (1) does not hold, then the algorithm BUFFER can start with the given allocation instead of using the initialization $B_i = 1$, for $1 \leq i \leq N$ in step a) above. In that case, the new buffer allocation $\{B'_i : 1 \leq i \leq N\}$ obtained by the algorithm satisfies the following conditions, and gives a minimal increment in $B'_i$'s so that the condition (1) is satisfied.

i) $B'_i \geq B_i$, for $i = 1, 2, ..., N$.
ii) $\{B'_i\}$ satisfies (1), and
iii) $\sum_{i=1}^{N} B'_i$, or equivalently, the total increase $\sum_{i=1}^{N} (B'_i - B_i)$ is minimum subject to (1).

Lemma 1. Let $\{B_i\}$ be any buffer allocation scheme which satisfies inequality (1). Let $j$ be the contact node of a terminal cycle $C$. Then there is another dfba $\{B'_i\}$ where each non-contact node of $C$ has unit buffer allocation, i.e., $B_i = 1$ for $i \not\in j$ and $i \in C$ such that $\sum B_i = \sum B'_i$.

Proof. Let $i$ be any node in $C$ and $i \not\in j$. If the buffer size $B_i$ at $i$ is greater than one, then decrease $B_i$ to $B'_i = 1$ and increase the buffer size at $j$ from $B_j$ to $B'_j = B_j + B_i - 1$. It is easy to see that $\{B'_i\}$ is still a dfba and $\sum B_i = \sum B'_i$. Repeat the above process for each $i \not\in j \in C$. This proves the lemma.
4. Example

We illustrate the algorithm by computing a set of $B_i$'s for the network in Figure 1 using $K = 6$ jobs. Each row in Table 1 below shows only the changes made to the buffer sizes in that step. In particular, step (e) is not shown.

![Figure 1. A network and a deadlock free buffer allocation for $K = 6$ jobs; next to each node is the value $B_i$.](image)

Table 1. Illustration of the algorithm for computing the deadlock free buffer allocation $\{B_i; 1 \leq i \leq n\}$ for $K = 6$; here $\beta = 27$.

<table>
<thead>
<tr>
<th>Step #</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
<th>$B_4$</th>
<th>$B_5$</th>
<th>$B_6$</th>
<th>$B_7$</th>
<th>$B_8$</th>
<th>$B_9$</th>
<th>$B_{10}$</th>
<th>$B_{11}$</th>
<th>$B_{12}$</th>
<th>$B_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b) $C = (1, 2, 3, 4)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) $C = (4, 5, 6)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) $C = (7, 8)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) $C = (8, 9, 10, 11)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) $C = (12, 13)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) $C = (5, 8, 12)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final values of $B_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
5. The Algorithm CYCLE for Cacti Networks

The step (b) of the algorithm for computing $\beta$ requires that a terminal cycle of the network be found. The algorithm CYCLE below finds all cycles in a cacti network. A terminal cycle can be then selected by first finding the number of contact nodes in the cycle. A node is a contact node if and only if its outdegree (number of arcs leaving that node) is greater than one. Since $\Gamma$ is a cactus network, and each arc of $\Gamma$ belongs to a cycle the outdegree of a node equals its indegree (number of arcs entering that node). When a cycle is deleted from the network in step (e) of the algorithm for $\beta$, the outdegree of each node in the cycle is decreased by one. This will allow the deletion of a terminal cycle subsequently. The algorithm takes linear time in the number of nodes in the network. The more general algorithm in Johnson [8] for finding all cycles, which works for all directed graphs and is also linear in the number of cycles and nodes, is considerably more complex. This prompted us to design the simpler algorithm CYCLE given here. The algorithm CYCLE is presented in a different format than the algorithm BUFFER in section 4.1 as it uses a more complex control flow and data structure to achieve its efficiency. The stack $S$ stores the nodes that have been visited but not yet outputted as part of a cycle; initially $S$ is empty. The array $\text{visit}[]$ is used for identifying the nodes which have been already visited, indicated by $\text{visit}[i] = 1$; initially, $\text{visit}[i] = 0$ for each $i$. The array $\text{d}[i]$ denotes the outdegree of the node $i$. The "startlist" is a list of visited contact nodes for which all cycles containing them have not been outputted and needs further processing. We assume that the network is represented as a list of adjacency lists, $\text{adj}[i]$, one list per node. The lists $\text{adj}[i]$ are gradually reduced to empty lists as items are deleted from them.
procedure CYCLE(N);
begin
  If (there is no node of d[i] > 1) then
    N is a cycle
  else begin
    choose a node i such that d[i] > 1;
    initialize startlist to i;
    visit[i] ← 1;
    while (startlist is not empty) do
      begin
        choose i from startlist and remove it from startlist;
        add i to S;
        while (S is not empty) do
          begin
            i ← top(S);
            if (adj[i] is non-empty) then
              begin
                choose the first node j in adj[i];
                remove j from adj[i];
                d[i] ← d[i] − 1;
                if (visit[j] = 1) then {a cycle is found at j}
                  begin
                    repeat {output of the cycle}
                      k ← top(S);
                      output k;
                      if (d(k) > 1) then
                        add k to the beginning of startlist;
                      pop(S);
                      until (top(S) = j);
                      output j;
                      end
                  else begin
                    visit[j] ← 1
                    push(j, S);
                    end;
              endif;
            else
              pop(S); {S is empty now}
            endwhile;
          end;
      endelse;
  end;
end.

The algorithm CYCLE is illustrated in Figure 2 using the network in Figure 1. We show the values of the stack S and the startlist as they change in the first few iterations of inner while loop, assuming that the algorithm is started at the contact node 4. The nodes of the cycles are outputted in the reverse order.
(i) The adjacency lists used in the computation shown in (ii)

<table>
<thead>
<tr>
<th>Node</th>
<th>Stack S</th>
<th>Startlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(4)</td>
<td>(4)</td>
</tr>
<tr>
<td>5</td>
<td>(5, 4)</td>
<td>empty</td>
</tr>
<tr>
<td>8</td>
<td>(8, 5, 4)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(12, 8, 5, 4)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>cycle = (12, 8, 5)</td>
<td>(5, 4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(8, 12)</td>
</tr>
<tr>
<td>6</td>
<td>(6, 5, 4)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cycle = (6, 5, 4)</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(1, 4)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(2, 1, 4)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(3, 2, 1, 4)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cycle = (3, 2, 1, 4)</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(8)</td>
<td>(12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ii) Part of the computation starting at node 4

Figure 2. Illustration of the algorithm CYCLE.
6. References

Throughput and Response Time Optimization in Queueing Network Models of Computer Systems

I. F. Akyildiz* and G. Bolch

*School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

**Informatik IV
University of Erlangen-Nuernberg
8520 Erlangen
F. R. G.

ABSTRACT

Queueing network models are being used to analyze various optimization problems such as server allocation, design and capacity issues, optimal routing and workload allocation in computer systems, communication networks and flexible manufacturing systems. This paper presents procedures for optimizing the performance and cost parameters for closed queueing network models with arbitrarily connected single and infinite server stations. The throughputs are maximized for fixed costs. The mean response times are minimized for given fixed throughput values. In both cases the linear and nonlinear cost functions are considered. The method of Lagrange multipliers is used to solve the optimization problems. Numerical examples are given to illustrate and to discuss the solutions.

Key Words: Computer System Design, Performance Evaluation, Optimization, Queueing Networks, Lagrange Multipliers.

1. Introduction

Cost evaluation and performance prediction processes are an important step in the planning and design of computer systems, communication networks and flexible manufacturing systems. Queueing network models have received special interest for performance analysis in the last two decades. Several analytical methods have been derived for the analysis of queueing network models in recent years [6, 11, 24]. In addition to the computation of performance meas-
ures, queueing network models can also be used for optimization of performance measures. In the optimization procedure an objective function, such as the costs, the throughput, the utilization or response time, can be obtained by appropriate selection of certain system input parameters which are called as decision variables. Within the modeling by queueing networks, the quantities such as service rates and the number of jobs can be assumed as decision variables by which the objective function is to be reached. Decision variables are selected subject to certain constraints. For example, we optimize the throughput of a given queueing network model by proper selection of service rates at each station subject to the cost constraints which restrict the range of the decision variables. Queueing network models represent the relationship between the objective function and the system input parameters. Within these models it is possible to have an optimal design of computer systems, communication networks and flexible manufacturing systems by using the mathematical optimization techniques.

Several authors have discussed the issue of optimization in recent years. Trivedi and Wagner [21] consider a computer configuration design problem where the computer system is modeled by a closed central server model. The system throughput is the objective function to be maximized by proper choice of device speeds subject to a cost constraint. A non-linear cost function is considered in the analysis. Trivedi, Wagner and Sigmon [22], Trivedi and Sigmon [23] analyze a computer system configuration problem in which the objective is to select the CPU speed, the capacities of secondary storage devices and the allocation of a set of files across the secondary storage devices so as to maximize the system throughput subject to a cost constraint. Kleinrock [10], Chandy, Hogarth and Sauer [5] consider a similar decision model as Trivedi and Kinicki [20] but their model is an open queueing network model. Chandy, Hogarth and Sauer [5] extend Kleinrock's model [10] in two different directions. First they allow locally balanced open queueing networks with multiple jobs classes as well
as the open networks analyzed by diffusion approximation [12]. Second they consider a rich class of nonlinear cost functions. Ferrari [8] uses a cyclic queueing model to optimize throughput subject to a nonlinear cost constraint. He solves the problem graphically and hence his method is restricted to problems with only a few devices.

Von Mayrhauser and Trivedi [25] consider a configuration design problem where the computer system is modeled as a closed queueing network. The mean response time to an interactive user request is minimized and the speeds of the devices are the decision variables. Geist and Trivedi [7] developed an optimization model for assigning a fixed set of files across an assemblage of memory devices so as to maximize system throughput. Trivedi and Kinicki [20] and Trivedi and Wagner [21] consider single server queueing networks and show by using the results of Price [15] that the optimization problem is global. They use the convolution algorithm as the base for optimization and maximize the throughput and minimize the costs. Note also that they do not give explicit closed form solutions for the optimization problem.

Kenevan and von Mayrhauser [9] show that the throughput is a log convex function of the number of items in a closed, single class, network of an arbitrary number of single and infinite servers. They also prove that reciprocal throughput is a convex function of the relative utilization of the servers which is the generalization of Price's proof [15]. Kobayashi and Gerla [13] determine the optimal routing in central server models with single server stations and multiple classes of jobs. Heiss and Totzauer [8] consider open BCMP queueing networks with load independent service rates. They determine the throughputs to optimize a linear objective function which is chosen to be a weighted sum of station utilizations. As restrictions, minimal and maximal throughputs and maximal response times are allowed per job class. Stecke [19] investigates BCMP networks in which she imposes a constraint on the total workload in the system. She shows that throughput as a function of the ratio of the service rate at
a server to the sum of the workloads, not purely concave but rather quasi-concave.

This paper is organized as follows: In section 2 the optimization problems are presented. In section 3 we briefly describe the mean value analysis, henceforth in short form MVA, and derive the constraint which will be used in application of Lagrange multiplier technique for optimization of performance measures. In section 4 we obtain an iterative formula for maximum throughput. The formula has a closed form solution in case of linear cost constraints. Response time minization procedure is given in section 5. Additionally, numerical examples are given to illustrate and to discuss the solutions.

2. Optimization

In the optimization procedure of performance measures such as response times, throughputs, utilizations, the objective is to determine the optimal service rates as decision variables subject to certain constraints. The cost constraint is the most common case and has the following form in the linear case:

\[ \sum_{i=1}^{N} \alpha_i \mu_i = C \]

where \( N \) is the number of stations in the queueing network. The parameter \( \alpha_i \) is the cost constraint for station \( i \). \( C \) denotes the total cost of the queueing network.

The nonlinear cost function is the more realistic case where the total costs are computed as follows:

\[ \sum_{i=1}^{N} \alpha_i \mu_i^{\alpha_i} = C \quad \alpha_i \geq 1 \]

where \( \alpha_i \) controls the increase of the costs for each station \( i \) in the network. For the special case, \( \alpha_i = 1 \) for \( i=1,2,\ldots,N \) we obtain the linear cost function, equation (1).

In the following we summarize the different optimization problems which we will investigate in this work.
i) Throughput Maximization under Fixed Costs

The total cost for the queueing network model is assumed to be known. The total cost is the budget available for purchasing a given number of stations with specific service rates. The total throughput is maximized by determining optimal service rates under cost constraints. We investigate this problem in section 4.

ii) Response Time Minimization for Each Station

The response time of each station for a given queueing network is minimized where a given fixed throughput value is controlled. The service rates are also, in this case, under cost constraints. In section 5 a solution is given to this problem.

3. Mean Value Analysis

We consider BCMP [3] queueing networks with \( N \) stations and \( K \) jobs. We denote \( \mu_i \) as the service rate of the \( i \)-th station. As generally known, BCMP networks contain the following station types: Type I: \( /M/m - FCFS \), Type II: \( /G/1 - PS \), Type III: \( /G/IS \) (Infinite Servers), Type IV: \( /G/1-LCFS-PR \).

The Bard/Schweitzer [2,18] algorithm provides the following solution for the analysis of BCMP networks containing single and infinite server stations.

The mean response time of the \( i \)-th station (for \( i = 1, \ldots, N \)) is:

\[
\overline{t}_i = \begin{cases} 
\frac{1}{\mu_i} (1 + \frac{K-1}{K} F_i) & \text{Type}(i) \neq IS \\
\frac{1}{\mu_i} & \text{Type}(i) = IS
\end{cases}
\]  

(3)

The throughput of the network is obtained by Little's law:

\[
\lambda = \frac{K}{\sum_{i=1}^{N} e_i \overline{t}_i}
\]  

(4)
The mean number of jobs at the $i$-th station (for $i = 1, \ldots, N$) is also obtained by Little's law:

$$E_i = \lambda \cdot e_i \cdot \bar{q}_i$$  \hspace{1cm} (5)$$

where $e_i$ is the mean number of visits that a job makes to station $i$:

$$e_i = \sum_{j=1}^{N} e_j \cdot p_{ij} \quad \text{for } i = 1, \ldots, N.$$  \hspace{1cm} (6)$$

$p_{ij}$ is the transition probability that a job after completing service at station $j$ proceeds to station $i$.

In the following we derive new formulas which provide a different perspective on performance measures like mean response times and mean number of jobs.

By substituting equation (5) in equation (3) we obtain:

$$\bar{q}_i = \begin{cases} \frac{1}{\mu_i} \left( 1 + K \frac{1}{K} \lambda e_i \bar{q}_i \right) & \text{Type}(i) \neq IS \\ \frac{1}{\mu_i} & \text{Type}(i) = IS \end{cases}$$  \hspace{1cm} (7)$$

Solving equation (6) for $\bar{q}_i$ we get

$$\bar{q}_i = \begin{cases} \frac{1}{\mu_i} \left( 1 - K \frac{1}{K} \lambda e_i \right) & \text{Type}(i) \neq IS \\ \frac{1}{\mu_i} & \text{Type}(i) = IS \end{cases}$$  \hspace{1cm} (8)$$

From equation (4) we derive

$$\lambda \cdot \sum_{i=1}^{N} e_i \bar{q}_i = K$$  \hspace{1cm} (9)$$

From equations (7 and 8) we obtain

$$\sum_{\text{Type}(i) \neq IS} \frac{\lambda e_i}{\mu_i} + \sum_{\text{Type}(i) = IS} \frac{\lambda e_i}{\mu_i} = K$$  \hspace{1cm} (10)$$
Equation (9) is the constraint used in application of Lagrange multipliers technique for optimization of performance measures. In [1] we assumed \( \frac{K-1}{K} \) is approximately equal to one and simplified equation (9). Using the simplified equations we then solved the optimization problems accordingly. However, the accuracy of the results is violated by this assumption. Similar way of attacking the optimization problems as in [1] was also utilized by [4,17].

4. Throughput Optimization for Fixed Costs

In the planning phase of computer systems, communication networks and flexible manufacturing systems we have to consider the fact that only a certain amount of money is available. The objective is to optimize the performance measures within the available budget. In this section we show how the throughput can be maximized by selecting the service rates of each station within the available budget.

The objective here is to find the optimal service rates \( \mu_i^* \) which provide the maximum throughput \( \lambda^* \) subject to the nonlinear costs. The linear cost is a special case of the nonlinear case. Note that another solution was given by Trivedi and Wagner [21] to this problem. They derive an explicit formula for total throughput \( \lambda \) from the normalization constant which is then maximized subject to the linear costs, equation (1), dependent on service rates. However, queueing networks studied by [21] may contain only single server stations. We include infinite server stations to the model considered for optimization problems. The solution suggested by [21] is further simplified here by using the MVA as the base for optimization.

The optimal service rates \( \mu_i^* \) which determine the maximum throughput \( \lambda \) for a given cost constraint cannot be computed by a closed form solution. However, they are determined iteratively as follows:

- **Initialize** the auxiliary quantities

\[
\mu_i^{(0)} = 1 \quad \text{for } i = 1, \ldots, N
\]
 Iterate for $n = 1, 2, ...$ until the deviation between the iterations is small:

$$
\lambda^{(s)} = \frac{C \, K}{\left( \sum_{i=1}^{N} \sqrt{c_i \, e_i \, \mu_i^{p-1(s)}} \right)^2 + (K-1) \sum_{i \neq IS} c_i \, e_i \, \mu_i^{p-1(s)}} \tag{10}
$$

$$
\mu_{t+1}^{(s)} = \begin{cases} 
\frac{\lambda^{(s)} \, e_t}{K} \left( \sum_{i=1}^{N} \sqrt{c_i \, e_i \, \mu_i^{p-1(s)}} \right)^2 + (K-1) \sum_{i \neq IS} c_i \, e_i \, \mu_i^{p-1(s)} 
& \text{for } Type(i) \neq IS \\
\frac{\lambda^{(s)} \, e_t}{K} \left( \sum_{i=1}^{N} \sqrt{c_i \, e_i \, \mu_i^{p-1(s)}} \right)^2 
& \text{for } Type(i) = IS 
\end{cases} \tag{11}
$$

Note that based on our test studies we found out that the iteration converges for $0 < \alpha < 2$. This is also based on the fact that the values for $\mu_i$ are not initiated appropriately.

**Derivation.**

First we rewrite equation (9) as follows:

$$
\sum_{\forall p(i) \neq IS}^{N} \frac{\lambda \, e_i}{\mu_i - \frac{K-1}{K} \lambda \, e_i} + \sum_{\forall p(i) = IS}^{N} \frac{\lambda \, e_i}{\mu_i} = K \tag{12}
$$

Equation (12) implicitly defines $\lambda$ as a function of $\mu$, i.e., $\lambda = \lambda(\mu)$

By differentiating equation (12) by $\mu_i$ we obtain

$$
\frac{\partial \lambda}{\partial \mu_i} = \begin{cases} 
\frac{\lambda \, e_t}{(\mu_i - \frac{K-1}{K} \lambda \, e_i)^2} \cdot A 
& \text{for } \neq IS \\
\frac{\lambda \, e_t}{\mu_i^2} \cdot A 
& \text{for IS} 
\end{cases} \tag{13}
$$

where

$$
A = \left( \sum_{j \neq IS}^{N} \frac{e_j \, \mu_j}{(\mu_j - \frac{K-1}{K} \lambda \, e_j)^2} + \sum_{j \neq IS}^{\mu} \frac{e_j}{\mu_j} \right)^{-1} \tag{14}
$$

The Lagrange function $L(\mu, y)$ with objective function $\lambda(\mu)$ is written as follows:

$$
L(\mu, y) = \lambda(\mu) + y \left( \sum_{i=1}^{N} c_i \, \mu_i^p - C \right) \tag{15}
$$

where $\lambda(\mu)$ is defined by equation (12).
By differentiating $L(\mu, y)$ by $\mu_i$ and $y$ we obtain the following nonlinear system equation for determining the maximum throughput and the optimal service rates:

$$\frac{\partial L}{\partial \mu_i} : \frac{\partial \lambda}{\partial \mu_i} + y \alpha \epsilon_i \mu_i^{\beta-1} = 0 \quad (16)$$

$$\frac{\partial L}{\partial y} : \sum_{i=1}^{N} \epsilon_i \mu_i^{\beta} = C \quad (17)$$

By substituting equation (13) into (16) we obtain

$$\frac{\partial L}{\partial \mu_i} = \begin{cases} \frac{\lambda \epsilon_i}{\mu_i - \frac{K-1}{K} \lambda \epsilon_i} \times A + y \alpha \epsilon_i \mu_i^{\beta-1} = 0 & \text{for } \not\in IS \\ \frac{\lambda \epsilon_i}{\mu_i} \times A + y \alpha \epsilon_i \mu_i^{\beta-1} = 0 & \text{for } IS \end{cases} \quad (18)$$

Rewriting equation (18) provides

$$\frac{\partial L}{\partial \mu_i} = \begin{cases} \frac{\lambda \epsilon_i}{\mu_i - \frac{K-1}{K} \lambda \epsilon_i} \times A + y \alpha \epsilon_i \mu_i^{\beta-1} (\mu_i - \frac{K-1}{K} \lambda \epsilon_i) = 0 & \text{for } \not\in IS \\ \frac{\lambda \epsilon_i}{\mu_i} \times A + y \alpha \epsilon_i \mu_i^{\beta-1} = 0 & \text{for } IS \end{cases} \quad (19)$$

By summing equation (19) over all stations $i$ we obtain

$$K \times A + y \alpha (C - \frac{K-1}{K} \lambda \sum_{\not\in IS} \epsilon_i \epsilon_i \mu_i^{\beta-1} \times A) = 0 \quad (20)$$

Then it follows that

$$y \alpha (C - \frac{K-1}{K} \lambda \sum_{\not\in IS} \epsilon_i \epsilon_i \mu_i^{\beta-1} \times A) = 0 \quad (21)$$

By substituting equation (21) into equation (18) we derive

$$\left[ \frac{K \lambda \epsilon_i \epsilon_i \mu_i^{\beta-1}}{C - \frac{K-1}{K} \lambda \sum_{\not\in IS} \epsilon_i \epsilon_i \mu_i^{\beta-1}} \right]^{1/2} = \begin{cases} \frac{\lambda \epsilon_i}{\mu_i - \frac{K-1}{K} \lambda \epsilon_i} & \text{for } \not\in IS \\ \frac{\lambda \epsilon_i}{\mu_i} & \text{for } IS \end{cases} \quad (22)$$

By summing over all stations $i$ we obtain

$$\left[ \frac{K \lambda}{C - \frac{K-1}{K} \lambda \sum_{\not\in IS} \epsilon_i \epsilon_i \mu_i^{\beta-1}} \right]^{1/2} \sum_{i=1}^{N} \sqrt{\epsilon_i \epsilon_i \mu_i^{\beta-1}} = K \quad (23)$$

By solving equation (23) for $\lambda$ equation (10) is derived.
From equation (23) we obtain

\[ C - \frac{K-1}{K} \lambda \sum_{i \neq IS} c_i e_i \mu_i^{p-1} = \frac{\lambda}{K} \left( \sum_{i=1}^{N} \sqrt{c_i e_i} \mu_i^{p-1} \right)^2 \]  

(24)

By substituting equation (24) into equation (22) and by rewriting we obtain equation (11).

Note that for linear costs, i.e., \( \alpha = 1 \), equation (10) and (11) become the following closed forms, i.e., there is no need for iterations in linear cost case.

\[ \lambda^* = \frac{C \cdot K}{\left( \sum_{i=1}^{N} \sqrt{c_i e_i} \right)^2 + (K-1) \sum_{i \neq IS} c_i e_i} \]  

(25)

\[ \mu_i^* = \begin{cases} \frac{\lambda e_i}{K} \left( \sum_{j=1}^{N} \sqrt{c_j e_j} \right) + (K-1) \sum_{j \neq i} \sqrt{c_j e_j} & \text{for Type}(i) \neq IS \\ \frac{\lambda e_i}{K} \sum_{j=1}^{N} \sqrt{c_j e_j} & \text{for Type}(i) = IS \end{cases} \]  

(26)

Example.

We consider the following queueing network model:

![Queueing Network Model](image)

Figure 1.

There are \( N = 5 \) stations, \( K = 20 \) jobs. Stations 4 and 5 are of Type 3 where other stations have Types of 1, 2 or 4. Transition probabilities are given by:

\[ p_{12} = 0.2; \ p_{14} = 0.5; \ p_{15} = 0.3; \ p_{43} = p_{21} = p_{31} = p_{51} = 1. \]
The mean number of visits \( e_i \) that a job makes to station \( i \) is:

\[
e_1 = 1; \ e_2 = 0.2; \ e_3 = 0.5; \ e_4 = 0.5; \ e_5 = 0.3.
\]

The service rate costs \( \zeta_i \) are:

\[
\zeta_1 = 10; \ \zeta_2 = 5; \ \zeta_3 = 5; \ \zeta_4 = 2; \ \zeta_5 = 1.
\]

We investigate the effect of different parameters \( \alpha \), of the cost function, equation (2), on the maximum throughput \( \lambda \). In Table 1 we show different service rates \( \mu_i^* \) computed using the iterative procedure, equations (10 and 11), for different \( \alpha \) values. We also give the throughput values \( (\lambda) \) which are obtained by mean value analysis using the optimal service rates \( \mu_i^* \).

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
<th>1.5</th>
<th>1.75</th>
<th>1.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_1^* )</td>
<td>37.153</td>
<td>12.688</td>
<td>7.242</td>
<td>5.105</td>
<td>4.011</td>
<td>3.358</td>
<td>3.330</td>
</tr>
<tr>
<td>( \mu_2^* )</td>
<td>7.810</td>
<td>2.689</td>
<td>1.550</td>
<td>1.105</td>
<td>0.801</td>
<td>0.748</td>
<td>0.678</td>
</tr>
<tr>
<td>( \mu_3^* )</td>
<td>19.154</td>
<td>6.642</td>
<td>3.738</td>
<td>2.630</td>
<td>2.078</td>
<td>1.745</td>
<td>1.563</td>
</tr>
<tr>
<td>( \mu_4^* )</td>
<td>1.085</td>
<td>0.496</td>
<td>0.370</td>
<td>0.331</td>
<td>0.320</td>
<td>0.321</td>
<td>0.326</td>
</tr>
<tr>
<td>( \mu_5^* )</td>
<td>1.226</td>
<td>0.551</td>
<td>0.405</td>
<td>0.359</td>
<td>0.344</td>
<td>0.343</td>
<td>0.347</td>
</tr>
<tr>
<td>( \lambda^* )</td>
<td>38.007</td>
<td>12.395</td>
<td>7.109</td>
<td>5.028</td>
<td>3.960</td>
<td>3.322</td>
<td>2.793</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>38.318</td>
<td>12.479</td>
<td>7.151</td>
<td>5.048</td>
<td>3.912</td>
<td>3.444</td>
<td>2.987</td>
</tr>
</tbody>
</table>

Table 1. (\( K = 70, \ C = 100 \))

The values for \( \alpha = 1 \) are computed from equations (25 and 26). From Table 1 it is clear that the service rates \( \mu_i \) decrease at the CPU with increasing \( \alpha \) since the cost factor is the greatest \( (\zeta_1 = 10) \) at the CPU. The following graph shows the relationship between the maximal throughput and the quantity \( \alpha \). The throughput decreases exponentially with increasing \( \alpha \).

Figure 2. Maximum Throughput dependent on \( \alpha \)
5. Response Time Minimization

The optimization question here is to minimize the objective function, the total response time $T$ subject to the fixed cost constraints. This is equivalent to the question to maximize the throughput since there is a dependency between the throughput and response time which can be seen in Little's law:

$$T = \frac{K}{\lambda}$$

From the equations

$$\lambda_i \bar{t}_i = \bar{c}_i$$
$$\lambda_i = \lambda c_i$$

for $i = 1, \ldots, N$ (28)

it is easy to show that the total response time $T$ can be computed by:

$$T = \sum_{i=1}^{N} e_i \bar{t}_i$$

(29)

We concentrate our investigation on maximizing the response times for different types of stations individually in the queueing network.

5.1. Single Server Stations

The objective is to minimize the response time $\bar{t}_j$ of a specific station of Type 1,2,4 where a given throughput value $\lambda$ is maintained. The service rates have a linear cost constraint, equation (1). The total cost $C$ is known. The response time $\bar{t}_j$ in a Type 1,2,4 station is given by Little's law:

$$\bar{t}_j = \frac{\bar{c}_j}{\lambda_j} = \frac{1}{\mu_j - \frac{K-1}{K} \lambda e_j}$$

(30)

If we minimize equation (30) as an objective function with conditions, equations (9 and 29) the following optimal service rates of the $j$-th station are obtained:

$$\mu'_j = \frac{-b + \sqrt{b^2 - 4a}}{2a}$$

(31)

for $j = 1, \ldots, N$ and $j \neq i$ with

$$a = K e_j$$

(32)

$$b = \lambda \left[ \sum_{i \neq j}^{N} \sqrt{e_i \bar{c}_i} \right]^2 - K \lambda e_j e_j - K C + (K-1) \lambda \sum_{i \neq j, \delta \neq i, j}^{N} e_i e_i$$

(33)

$$c = K \lambda e_j C - (K - 1) \lambda^2 e_j \sum_{i \neq j, \delta \neq i, j}^{N} e_i e_i - \frac{K-1}{K} \lambda^2 e_j \left( \sum_{i \neq j}^{N} \sqrt{e_i \bar{c}_i} \right)$$

(34)
The service rates of the remaining stations \( i \) for \( i = 1, \ldots, N \) and \( i \neq j \) are obtained by:

\[
\mu_i^* = \begin{cases} 
\Phi \sqrt{\frac{e_i}{c_i}} + \frac{K-1}{K} \lambda e_i & \text{for } \text{Type}(i) \neq \text{IS} \\
\Phi \sqrt{\frac{e_i}{c_i}} - \frac{1}{N} \sum_{i \notin j, \ j \neq \text{IS}} e_i e_i - e_j \mu_j^* & \text{for } \text{Type}(i) = \text{IS}
\end{cases}
\] (35)

where

\[
\Phi = \frac{C - \lambda \frac{K-1}{K} \sum_{i \notin \text{IS}} e_i e_i}{\sum_{i \notin j} \sqrt{e_i e_i}}
\] (36)

**Derivation.**

The Lagrange function is derived as follows:

\[
L(\mu, y_1, y_2) = \frac{1}{\mu_j - \frac{K-1}{K} \lambda e_j} + y_1 \left[ \sum_{i \notin \text{IS}} \lambda e_i \mu_i - \frac{K-1}{K} \lambda e_i \right] + \sum_{i \notin \text{IS}} \mu_i e_i - C
\] (37)

Differentiating by \( \mu_i, \mu_j, y_1 \) and \( y_2 \) we obtain the following system of equations:

\[
\frac{\partial L}{\partial \mu_j} = \frac{-1}{(\mu_j - \frac{K-1}{K} \lambda e_j)^2} + y_1 \left( \frac{-\lambda e_j}{(\mu_j - \frac{K-1}{K} \lambda e_j)^2} \right) + y_2 e_j = 0
\] (38)

for \( i, j = 1, 2, \ldots, N \) and \( i \neq j \).

\[
\frac{\partial L}{\partial \mu_i} = \begin{cases} 
y_1 \frac{-\lambda e_i}{(\mu_i - \frac{K-1}{K} \lambda e_i)^2} + y_2 e_i = 0 & \text{for } i \notin \text{IS} \\
y_1 \frac{-\lambda e_i}{\mu_i^2} + y_2 e_i = 0 & \text{for } \text{IS}
\end{cases}
\] (39)

for \( i, j = 1, \ldots, N \) and \( i \neq j \).

\[
\frac{\partial L}{\partial y_1} : \sum_{i \notin \text{IS}} \frac{\lambda e_i}{\mu_i - \frac{K-1}{K} \lambda e_i} + \sum_{i \notin \text{IS}} \frac{\lambda e_i}{\mu_i} = K
\] (40)

\[
\frac{\partial L}{\partial y_2} : \sum_{i = 1}^{N} e_i \mu_i = C
\] (41)
We can solve equation (39) as follows:

\[
\sqrt{\frac{y_2}{y_1}} \lambda e_i = \begin{cases} 
\frac{\lambda e_i}{\mu_i - \frac{K-1}{K} \lambda e_i} & \text{for } \notin IS \\
\frac{\lambda e_i}{\mu_i} & \text{for } IS 
\end{cases} 
\]  

for \( i,j = 1, \ldots, N \) and \( i \neq j \).

Rewriting equation (42) we get

\[
\mu_i = \left\{ \begin{array}{ll}
\sqrt{\frac{y_2}{y_1}} \lambda e_i + \frac{K-1}{K} \lambda e_i & \text{for } \notin IS \\
\sqrt{\frac{y_1}{y_2}} \lambda e_i & \text{for } IS 
\end{array} \right.
\]  

for \( i,j = 1, \ldots, N \) and \( i \neq j \).

By substituting equation (42) in equation (40) we obtain

\[
\frac{\lambda e_i}{\mu_j - \frac{K-1}{K} \lambda e_j} + \sqrt{\frac{y_2}{y_1}} \lambda \sum_{i \neq j} \sqrt{e_i e_j} = K
\]  

for \( i,j = 1, \ldots, N \) \( i \neq j \).

We substitute equation (43) in equation (41) and get

\[
e_j \mu_j + \sqrt{\frac{y_1}{y_2}} \lambda \sum_{i \neq j} \sqrt{e_i e_j} + \lambda \sqrt{\sum_{i \neq j \neq IS} e_i e_j} = C
\]  

(45)

For the sake of simplicity we introduce the following auxiliary quantities:

\[
\Phi = \sqrt{\frac{y_1}{y_2}} \lambda
\]  

(46)

\[
w = \sum_{i \neq j} \sqrt{e_i e_j}
\]  

(47)

\[
s = \sqrt{\sum_{i \neq j \neq IS} e_i e_j}
\]  

(48)

From (45) we obtain then equation (38) which has the following rewritten form

\[
\Phi = \frac{C - \lambda s - e_j \mu_j}{w}
\]  

(49)
We substitute (49) into (44) and obtain the following equation for determining the optimal service rates $\mu_i$ in the $j$-th station:

$$K e_i + \left(\lambda w^2 - K e_i K \epsilon_j - K e_j + (K-1) \lambda \epsilon_i \right) \mu_j + \lambda \epsilon_j \{K e_i - (K-1) \lambda \epsilon_i \} - \frac{(K-1)}{K} \lambda w^2 = 0 \quad (50)$$

Note that this equation generally possesses two solutions. The mean response times $\bar{t}_j$ are minimized when the optimal service rates $\mu_j$ are chosen to be the greater values of the solution. The lesser values maximize the response times. Using $\mu_j$ we determine the auxiliary variable $\Phi$ from equation (49) and the service rates $\mu_i$ of the remaining stations $i=1,..,N$ and $j \neq i$ from equation (35).

Remark. Equation (35) may not always admit a unique solution. This is the case when the desired throughput value $\lambda$ cannot be obtained under the given total cost $C$. In this case the solution procedure, equations (35 and 36), provides negative service rates which will be demonstrated in the following example. In the following example we minimize the response time at the CPU.

Example.

There are $N=4$ stations and $K=10$ total number of jobs. The total cost is $C=10$.

![Diagram of the system](image)

The mean number of visits $e_i$ is computed:

$$e_1 = 0.4 \quad ; \quad e_2 = 0.4 \quad ; \quad e_3 = e_4 = 0.3$$

Single costs for the service rates are $c_1 = c_2 = c_3 = c_4 = 1$

The objective is to minimize the response time $\bar{t}_1$ of the first station subject to the total cost $C=10$ while keeping the total throughput fixed at $\lambda=100$. 

---

-255-
From equations (31 and 35) we obtain the following optimal result for service rates:

\[ \mu_1^* = 108.49 \; ; \; \mu_2^* = -21.7 \; ; \; \mu_3^* = -23.41 \; ; \; \mu_4^* = -53.41 \]

It can easily be seen that the total throughput of \( \lambda = 100 \) cannot be reached for the given cost constraint of \( C = 10 \).

Now we select new total throughput as \( \lambda = 1 \) and obtain as optimal service rates

\[ \tilde{\mu}_1 = 0.122 \; ; \; \mu_1^* = 0.068 \; ; \; \mu_2^* = 0.471 \; ; \; \mu_3^* = 0.366 \; ; \; \mu_4^* = 0.006 \]

Using MVA with these optimal service rates we obtain the throughput and the mean response time of the first station:

\[ \lambda = 1.029 \; \; \; \tilde{\tau}_1 = 0.124 \]

5.2. Infinite Server Stations

Since terminals can be modeled as an infinite server station we investigate the case where the response time of Type 3 stations is minimized. The objective is to determine the optimal service rates \( \mu_j^* \) such that the response time \( \tilde{\tau}_j \) of a station \( j \) of Type 3 is minimized while reaching the value of given total throughput value \( \lambda \).

The objective function \( \tilde{\tau}_j \) is:

\[ \tilde{\tau}_j = \frac{1}{\mu_j} \]  \hspace{1cm} (51)

The optimal service rates \( \mu_j^* \) are computed by equation (31) with \( \Phi \) determined by equation (36). The parameter \( a \) which occurs in equation (31) is obtained from equation (32). However, the parameters \( b \) and \( c \) have a slightly different form here than in equations (33) and (34):

\[ b = \lambda \left[ \sum_{i \neq j}^{N} \sqrt{e_i} c_i \right]^2 - \lambda e_j e_i - K C + (K-1) \lambda \left[ \sum_{i \neq j}^{N} e_i c_i \right] \]  \hspace{1cm} (52)

\[ c = \lambda e_j \left[ C - \frac{K-1}{K} \lambda \sum_{i \neq j}^{N} e_i c_i \right] \]  \hspace{1cm} (53)
The Lagrange function \( L \) also has a different form than in equation (37):

\[
L(\mu, y_1, y_2) = \frac{1}{\mu_j} + y_1 \left\{ \sum_{\nu \neq i} \frac{\lambda e_i}{\mu_i - \frac{K-1}{K} \lambda e_i} + \sum_{\nu} \frac{\lambda e_i}{\mu_i} - K \right\} + y_2 \left[ \sum \mu_i c_i - C \right] \quad (54)
\]

Differentiating by \( \mu_i, \mu_j, y_1 \) and \( y_2 \) we obtain the following system of equations:

\[
\frac{\partial L}{\partial \mu_j} = -\frac{1}{(\mu_j)^2} + y_1 \frac{-\lambda e_j}{(\mu_j)^2} + y_2 \epsilon_j = 0 \quad (55)
\]

\[
\frac{\partial L}{\partial \mu_i}, \frac{\partial L}{\partial y_1} \quad \text{and} \quad \frac{\partial L}{\partial y_2}
\]

are given by equations (39, 40 and 41), respectively.

Equation (44) becomes

\[
\frac{\lambda e_j}{\mu_j} + \sqrt{\frac{y_2}{y_1}} \lambda \sum_{i \neq j} \sqrt{\epsilon_i \epsilon_j} = K \quad (56)
\]

Substituting equation (49) into equation (56) we obtain the following system of equations by which the optimal service rates \( \mu_j \) in the \( j \)-th station can be determined:

\[
K \epsilon_j \mu_j^2 + \left( \lambda \epsilon_j^2 - K \lambda \right) \epsilon_j - \lambda \epsilon_j \epsilon_j \mu_j + \lambda \epsilon_j \left( C - \frac{K-1}{K} \lambda \right) = 0 \quad (57)
\]

Example.

We investigate the same example as in Figure 3. We minimize the response time for fixed total cost of \( C = 10 \) while a total throughput \( \lambda = 1 \) is maintained. From equations (31 and 35) we obtain the optimal service rates.

\[
\mu_1^* = 1.119; \mu_2^* = 0.498; \mu_3^* = 0.390; \mu_4^* = 7.993
\]

Using the optimal service rate \( \mu_4^* \) we then compute the minimum response time for station 4 from equation (51)

\[
\bar{T}_4 = 0.125.
\]

Using these optimal service rates we run MVA and obtain the following throughput and mean response time results:

\[
\lambda = 1.004 \quad \bar{T}_4 = 0.125
\]

Let us also analyze the case where the service rates are modified and show the effect of this modification on throughput \( \lambda \) and response time \( \bar{T}_4 \). The total
Cost is still $C=-10$. Here we select a faster service time for terminal (station 4).

$$\mu_1 = 1 ; \mu_2 = 0.5 ; \mu_3 = 0.5 ; \mu_4 = 8$$

Using these service rates we compute by MVA

$$\lambda = 0.962 \quad \bar{t}_4 = 0.125$$

In this case the mean response time is the same as the above case. However, the throughput value decreases below the given throughput value of $\lambda = 1$.

Now we select small values for service rates of stations 1, 2 and 3 as

$$\mu_1 = 1.2 ; \mu_2 = 0.5 ; \mu_3 = 0.5 ; \mu_4 = 7.8$$

In this case we obtain with MVA

$$\lambda = 1.081 \quad \bar{t}_4 = 0.128$$

Let us select the following values for service rates which deviate more from the optimal values

$$\mu_1 = 1.5 ; \mu_2 = 0.8 ; \mu_3 = 0.7 ; \mu_4 = 7$$

Using these service rates MVA provides

$$\lambda = 1.451 \quad \bar{t}_4 = 0.142$$

As can easily be seen the total throughput is increasing for smaller values of service rates in stations 1, 2 and 3. Consequently, the mean response time $\bar{t}_4$ increases.

References


Selecting Capacities in Cyclic Networks of Queues with Finite Buffers

I. F. Akyildiz and Wei Liu
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

ABSTRACT

Solutions are presented for optimal buffer allocation problem in cyclic closed queueing networks with blocking. The problem is to distribute the given total capacity of the network to each station such that the network throughput will be maximized.

Key Words: Performance Evaluation, Queueing Networks, Blocking, Buffer Capacity.

1. Introduction

Since in actual systems the resources have finite capacity, queueing networks with blocking must be used for performance analysis. In queueing networks with blocking, a station can be thought of as a device with a finite length queue. Blocking arises because of limitations imposed by the capacity of these stations. However, blocking events cause interdependency between stations which makes, of course, the analysis very difficult. In recent years there has been a growing interest in the development of computational methods to analyze queueing networks with blocking. Exact results for blocking networks exist for very few special cases, for example, reversible networks such as two-station networks [1]. Other studies of blocking networks are given in [3,4,5,8,12,14]. A survey on blocking networks is given by Onvural [12] and Perros [13]. A lot of work has been done for production lines [6,7,9,11].

Production lines are usually modelled as open tandem networks of queues where the first station has an infinite capacity (accordingly arrivals cannot be lost) and the last station cannot be blocked. In general, the best way to allocate buffers in open tandem network models of production lines is to emphasize the queues in the middle. This phenomenon is known in the literature as the "bowl effect" [9,11].
[6] and Altiok/Stidham [7] consider a tandem line of servers with finite capacity buffers and independent and exponentially distributed service, up and down times. They maximize the average output rate by allocating the buffer capacities optimally.

The problem of buffer allocation is an important one, and as far as we know it has not been tackled within the context of closed queueing networks with blocking before. What we attack in this work is to distribute the given total capacity of the network to each station such that an optimal throughput will be achieved. It will be seen that in cyclic networks of finite queues the "bowl effect" does not exist because there is no open end and because all stations are equal (or symmetric) with respect to their position in the network. Therefore, the allocation of buffers in closed cyclic networks is different from the past studies of allocating buffers in open tandem network models of production lines.

The contribution of this work is:

i) Finding necessary and sufficient conditions for optimal capacity allocation in two-station networks.

ii) Finding a heuristic solution for distributing the capacities optimally in cyclic networks.

2. Model Description

The network contains \( N \) stations and \( K \) total number of jobs. Each station contains \( m_i \geq 1 \) servers with exponentially distributed service times.

Figure 1. Cyclic network of queues

Each station has a fixed finite capacity \( B_i \) where \( B_i = \text{(queue capacity} + m_i) \), (for \( i = 1, 2, \ldots, N \)). A job which is serviced by the \( i \)-th station proceeds to the \( j = (i+1) \mod N \)-th station, if the \( j \)-th station is not full, i.e., if the number of jobs in the \( j \)-th station, \( k_j < B_j \). Otherwise, the job is blocked in the server of the \( i \)-th station until another job in the \( j \)-th station has completed its servicing and a place becomes available. Furthermore, it is valid that
which means that the total number of jobs \( K \) in the network cannot exceed the total station capacity of the entire network. The service discipline in each station is First Come First Served.

Finite station capacities and blocking can introduce deadlock in queueing networks with transfer blocking. Deadlock may occur if a job which has finished its service at station \( i \)'s server wants join station \( j \), whose capacity is full. That job is blocked in station \( i \). Another job which has finished its service at \( j \)-th station now wants to proceed to the \( i \)-th station, whose capacity is also full. It blocks the \( j \)-th station. Both jobs are waiting for each other. As a result a deadlock situation arises.

A closed queueing network with finite station capacities is deadlock free [2], if and only if for each cycle in the network the following condition holds:

\[
K < \sum_{j \in C} B_j
\]

Simply stated, the total number of jobs in the network must be smaller than the sum of station capacities in each cycle. Since cyclic queueing networks have only one cycle, this condition, equation (2) corresponds to equation (1). Equation (1) is a sufficient condition for cyclic networks to be deadlock free.

3. Two Station Networks

Akyildiz [1] showed that the state space of a blocking two-station queueing network is isomorphic to the state space of a nonblocking two-station network with appropriate total number of jobs \( \hat{K} \) which is computed by

\[
\hat{K} = \min(B_1 + m_1, K) + \min(B_2 + m_2, K) - K
\]

State spaces of both networks have the same structure, it follows that Markov processes describing the evolution of both networks have the same structure. This guarantees that the throughputs of both networks are exactly equal.

\[
\lambda_B(K) = \lambda_{NB}(\hat{K})
\]

The method of Akyildiz [1] shows that throughput can be maximized by maximizing \( \hat{K} \), i.e., the
number of jobs in the corresponding nonblocking two-station network. This is based on the fact that the throughput of closed queueing networks without blocking is an monotonically increasing function dependent on the total number of jobs. Also it is obvious that $\hat{K}$ is an increasing function dependent on the total number of states of the network. Therefore, it is sufficient to distribute the capacities in such a way that the total number of states in the network will be maximized.

Now suppose that the total capacity $B = B_1 + B_2$ is fixed. The problem is to determine $B_1$ and $B_2$ such that the total throughput $\lambda_B(K)$ of the blocking two-station network will be maximum.

- **Case 1:** If $B = K$, deadlock will occur [2] which is irrelevant to our study.

- **Case 2:** If $B < K$, the capacity is not sufficient to hold all the jobs. This case is impossible.

- **Case 3:** If $B \geq 2 \cdot K$, we distribute the capacities such that the condition, $B_1 \geq K$ and $B_2 \geq K$, is satisfied. In this case, the network becomes a nonblocking network.

- **Case 4:** If $K < B < 2 \cdot K$, the distribution of the capacities will influence the network throughput. The solutions are given in the following:

As generally known, the throughput is a nondecreasing function of the number of jobs in nonblocking closed queueing networks. The maximum $\hat{K}_{\text{max}}$ will provide the maximum throughput for the blocking network. Therefore, the necessary and sufficient conditions for maximizing the network throughput can be derived from equation (3).

**Condition 1:**

If $B \geq \{ 2 \cdot K - m_1 - m_2 \}$, the capacity is distributed to each station such that the conditions $(B_1 + m_1 \geq K)$ and $(B_2 + m_2 \geq K)$ are satisfied.

**Condition 2:**

If $K < B < \{ 2 \cdot K - m_1 - m_2 \}$, the capacity is allocated to each station such that the conditions $(B_1 + m_1 \leq K)$ and $(B_2 + m_2 \leq K)$ are satisfied.
Proof. We show that when the above conditions are satisfied, $\hat{K}$ reaches the maximum value $K_{\text{max}}$. If these conditions are not satisfied, $\hat{K}$ will be less than $K_{\text{max}}$ and the throughput will not be maximum.

First we consider Condition 1.

If $B \geq (2 \cdot K - m_1 - m_2)$ and $(B_1 + m_1) \geq K$ and $(B_2 + m_2) \geq K$, it follows from (3) that $K_{\text{max}} = K$.

Without loss of generality, suppose $(B_1 + m_1) < K$, then from (3) it follows that

$$\hat{K} = (\min(B_1 + m_1, K) + \min(B_2 + m_2, K) - K) \leq (B_1 + m_1 + K - K) = (B_1 + m_1) < K.$$ Thus, it follows that $\hat{K} < K_{\text{max}}$.

Similarly, we prove Condition 2:

If $B < (2 \cdot K - m_1 - m_2)$ and $(B_1 + m_1) \leq K$ and $(B_2 + m_2) \leq K$ then $K_{\text{max}} = (B + m_1 + m_2 - K)$.

Without loss of generality, suppose $(B_1 + m_1) > K$, then $(B_2 + m_2) < K - 1$. From (3) we obtain $\hat{K} = (K + B_2 + m_2 - K) = (B_2 + m_2)$. Since $(B_1 + m_1) > K$, it follows that $(B_1 + m_1 + B_2 + m_2 - K) > (K + B_2 + m_2 - K)$.

i.e., $(B + m_1 + m_2 - K) > (B_2 + m_2)$. Therefore, $\hat{K} < K_{\text{max}}$.

4. Networks with $(N > 2)$ Stations

Based on the same state space transformation concept as mentioned in section 3, Akyildiz [3,5] introduced an approximate method for the computation of throughput values in blocking networks with more than two stations. The idea is to find a corresponding nonblocking queueing network with appropriate total number of jobs $\hat{K}$ such that the number of states in both networks will be equal or at least approximately equal. This would imply that both networks have approximately the same stochastic behavior and the throughputs of both networks are approximately equal.

$$\lambda_B(K) = \lambda_{NB}(\hat{K}) \quad (4)$$
The total number of jobs $\hat{K}$ in the nonblocking network is computed such a way that it will provide the same number of states as in the blocking network. Akyildiz [3,5] gives an efficient convolution algorithm for the computation of the feasible states in blocking networks.

We use the same arguments as before that $\hat{K}$ is an increasing function of the total number of states in the network. By maximizing the number of states we can achieve an optimal allocation of the capacities.

**Case 1:** If $B = K$, then the network will have deadlock which is irrelevant to our study.

**Case 2:** If $B < K$, the capacity is not sufficient to hold all the jobs. Thus, it is an impossible case.

**Case 3:** If $B \geq (N \cdot K)$, we can distribute the capacity to all stations such that the conditions $\{ B_i \geq K \}$ (for $i = 1,2,..N$) are satisfied. In this case, the network becomes a nonblocking network.

**Case 4:** If $K < B < (N \cdot K)$, the total number of states varies with the distribution of the capacity. The condition for optimal capacity distribution is as follows:

If $\{ B \geq 2 \cdot N \} \& \{ K = B - 1 \}$, the optimal capacity distribution is then $B_i \geq 2$ (for $i = 1,2,..,N$).

In all other cases, the distribution of the capacity is executed such that the condition $|B_i - B_j| \leq 1$, for $i, j = 1,2,..,N$, will be satisfied. In other words, the total capacity will be distributed among the stations uniformly.

Any capacity distribution which satisfies these conditions, will provide the maximum throughput in cyclic blocking networks. This is verified by executing several examples.

5. Conclusion

In this paper, we presented solutions for optimal distribution of capacities in cyclic queueing networks with blocking. The solutions are exact for two station networks and approximate for networks
with more than two stations. The problem becomes complicated in case of networks with arbitrarily connected stations. One possible solution is to allocate a minimum number of capacities ($B_{\text{min}}$) to guarantee that deadlock will not occur, satisfying equation (2). The remaining capacities ($B - B_{\text{min}}$) can be distributed in such a way that the common stations (i.e., stations occur in more than one cycle in the network) receive more capacities than the other stations. This idea needs to be verified and extensively studied.
References

Product Form Approximations for Queueing Networks with Multiple Servers and Blocking

I. F. AKYILDIZ, MEMBER, IEEE

Abstract—Queueing networks with blocking have become an important research topic in performance evaluation in recent years. Various types of blocking within queueing networks have been studied by several investigators. In this work, the following type of blocking is examined. Upon completing of its service, a job attempts to enter a new station. If at that moment, the destination station is full, the job is forced to reside in the server of the source station until a place becomes available in the destination station. The server of the source station remains blocked during this period of time. It is shown that the equilibrium state probabilities for this type of blocking queueing network have an approximate product form solution. The solution is based on the concept of normalizing the feasible states that violate station capacities. The states are adapted to the allowed capacities of the stations. However, the equilibrium state probabilities allow only the computation of mean number of jobs. In order to obtain the throughput values, the concept of the state space transformation is introduced. This concept is based on finding a nonblocking network with appropriate total number of jobs of which the number of feasible states is equal or approximately equal to the number of feasible states in the blocking queueing network. This guarantees that the Markov processes describing the evolution of both networks over time have approximately the same structure. This leads to the result that the throughput of both systems are approximately equal. The approximations are validated by executing several examples, including stress tests, and comparing them with simulation results.

Index Terms—Binomial coefficient formula, blocking, finite station capacities, performance evaluation, performance measures, queueing networks.

I. INTRODUCTION

A COMPUTER system can be viewed as an organized collection of hardware and software resources for which the concurrent processes in the system are constantly competing. Two major functions of an operating system are the effective scheduling of conflicting requests and the appropriate handling of the process queues waiting for resource allocation and scheduling. Queueing network theory can be used to provide the basic framework and mathematical tools for the modeling and analysis of computer system hardware and software.

Queueing networks have received a very special interest in the last 15 years for uses with performance evaluation and performance prediction. A queueing network is a collection of stations (devices with queues) in which jobs/processes proceed from one station to another in order to satisfy their service requirements. When dealing with queueing networks with infinite station capacities, Baskett et al. [10] and Kelly [21] have shown that the solutions for four types of stations (*/M/M/FCFS, */G/1-RR-PS, */G/I5 (Infinite Servers), and */G/I-LCFS-PR) have a product form. The product form implies that each station in the network can be analyzed independently from each other. Several algorithms have been introduced for an efficient computation of performance measures for queueing networks in the last 15 years [27], [33], [34], [35]. In the last two decades, many successful applications have been developed for the modeling of computer, communication, and manufacturing systems by queueing networks [27], [35], [40]. Since in actual systems the resources have finite capacity, queueing networks with blocking must be used for performance analysis. In queueing networks with blocking, a station can be thought of as a device with a finite length buffer. Blocking arises because of the limitations imposed by the capacity of these buffers.

In recent years, there has been a growing interest in the development of computational methods to analyze queueing networks with blocking. Researchers from various areas such as computer science, operations research, mathematics, and electrical engineering have studied blocking networks. Several papers have been published dealing with various types of blocking types. Previous work regarding the blocking networks falls into three classes [28].

1) Transfer (Type 1 = Manufacturing) Blocking: Upon completion of its service, a job at station $i$ attempts to enter destination station $j$. If station $j$ is full at that moment, the job is forced to wait at server $i$ until it enters destination station $j$. The server remains blocked for this period of time. It cannot serve any other job waiting in the queue. This type of blocking has been used to study models of flexible manufacturing systems and disk I/O subsystems [1]–[5], [8], [12], [14], [29]–[31], [36], [38], [39], [41].

2) Service (Type 2 = Communication...I) Blocking: A job in station $i$ declares its destination station $j$ before it starts its service. If station $j$ is full, the $i$th server is blocked, i.e., it cannot serve jobs. When a departure occurs from destination
station j, the ith server becomes unblocked and the job begins receiving service. This blocking type has been used to study models of production lines and telecommunication networks [11], [14], [16], [17], [22], [23], [25], [37].

3) Rejection (Type 3 = Communication2) Blocking: Upon service completion at station i, a job attempts to join destination station j. If station j is full at that moment, the job is refused. The rejected job goes with a certain probability (so-called "rejection probability") back to the station i's server and immediately receives a new service with the same mean service time. This is repeated until some job completes a service at station j and a place becomes available. This blocking type has been used to model communication networks [6], [7], [9], [13], [15], [18], [19], [21], [26], [32], [42].

Comparisons between these distinct types of blocking have been carried out by Onvural and Perros [28]. Several other investigators in recent years have published results on blocking queueing networks. Since we investigate closed queueing networks with transfer blocking here, the discussion of the previous work will contain the studies of these networks.

The analysis of transfer blocking networks is very difficult. Exact results exist only for very limited cases. In [1], we showed that the equilibrium state probability distributions of two-station closed queueing networks with transfer blocking and multiple server stations are identical to those of two-station closed queueing networks without blocking. Onvural and Perros [29] show that if the number of jobs in a network with transfer blocking is one more the capacity of the station with the smallest capacity, there exists an exact product form solution.

All other studies for this type of networks are approximations. In [3], we showed that the throughput of a blocking network with K total number of jobs and single servers is approximately equal to the throughput of a nonblocking network with an appropriate total number of jobs K. The approximation provides very accurate results, on the average 5 percent deviations. In [4], we modify mean value analysis, in short form MVA [34], for transfer blocking networks. The resulting method is called MVABLO. MVABLO estimates the additional time a job spends in each station because of blocking. This time is included in the response time formula of MVA. Even though the method is extremely fast, it provides larger deviations from actual values. These results have been extended to networks with general service time distributions in [2].

Suri and Diehl [38] consider transfer blocking policy in cyclic networks. They present an approximate method to compute the throughput of the network. They approximate groups of two stations by a variable capacity station, defined as a superposition of fixed capacity stations. They start with the last two stations and successively reduce the network until two stations in tandem remain. The method is easy to implement and shows good accuracy but involves much computation. At each step, all conditional probabilities have to be found, since they are used to construct the equivalent variable capacity station. The method only gives the throughput of the entire network, it does not give statistics for individual stations. Another disadvantage is that it is applicable only on cyclic networks where one of the stations must have an infinite capacity. Another drawback is that the capacity of each downstream station must be smaller than the total number of jobs in the network.

Perros, Nilsson, and Liu [30] give an algorithm for an arbitrarily connected network where some (very few) stations may have finite capacity. They partition the set of stations in a so-called blocking subnetwork and a nonblocking subnetwork. The nonblocking subnetwork containing infinite capacity stations is replaced by a composite station using parametric analysis for infinite capacity networks. The reduced network is then analyzed numerically. However, if all stations of the network have finite capacity this method reduces itself to a numerical analysis method which, as generally known, is only applicable on very small networks.

Deadlocks are possible in transfer blocking networks. All stations in a directed cycle could be full at one time. If in each of the stations of the cycle, the blocked job is scheduled to go to the next station in the cycle, the network is deadlocked. There are two possible solutions to the deadlock problem in blocking networks.

1) Include a strategy to handle deadlocks in the model. Perros, Nilsson, and Liu [30] assume that in case of deadlock all jobs involved move simultaneously to their destinations. This complicates the model, since the deadlock handling method influences the balance equations.

2) Simply restrict yourself to cases where deadlock is impossible. One such case arises whenever the number of jobs in the system is less than the capacity of the directed cycle with minimal capacity. No directed cycle can ever have all its stations full at the same time, and deadlock is impossible.

This paper is organized as follows: In Section II, we show that the equilibrium state probabilities for multiple server queueing networks with transfer blocking can be computed approximately. The solution is based on the concept of normalizing the infeasible states that violate station capacities. The states are adapted to the allowed capacities of the stations. However, the equilibrium state probabilities allow only the computation of the mean number of jobs. In order to obtain the throughput values, the concept of a state space transformation is introduced in Section III. In Section IV, two examples are given to explain both algorithms in the case of tandem and nontandem networks. Additional examples are given to compare our results to the results of Suri and Diehl in the evaluation Section V. In the Appendix, several examples are listed and the results are compared to simulation.

II. Approximate Product Form Solution

We consider closed queueing networks with the following assumptions.

1) There are N stations.
2) The number of jobs in the system is fixed at K. All jobs belong to the same class.
3) Each station may have multiple servers (m_i ≥ 1) and an exponentially distributed service time with mean value 1/μ_i (i = 1, 2, · · · , N).
4) Each station has a fixed finite capacity M_i (i = 1, 2, · · · ,
5) A job serviced by the $i$th station proceeds to the $j$th station with probability $p_{ij}$ for $i = 1, 2, \ldots, N$, if the number of jobs in the $j$th station has not exceeded the capacity $M_j$. Otherwise, the job is blocked in the $i$th station until a job in the $j$th station is serviced and a place becomes available.

6) The total number of jobs $K$ must be smaller than the sum of the station capacities, that is,

$$K < \sum_{i=1}^{N} M_i. \tag{1}$$

7) The service discipline in each station is first-come-first-served.

8) The network must be deadlock free. Deadlock occurs in blocking networks, if, for instance, assume that station $i$ is blocked by station $j$. Now it is possible that a job in station $j$ may, upon completion of its service, choose to go to station $i$. If station $i$ is full at that time, the deadlock will occur. The blocking network is deadlock free, if

$$K < \sum_{j \in C} M_j, \tag{2}$$

i.e., the total number of jobs in the network must be smaller than the sum of station capacities in each cycle $C$. Since tandem networks have only one cycle, this condition, (2), corresponds to (1). Equation (1) is a sufficient condition for tandem networks to be deadlock free.

9) If there are several stations all linked to station $j$, then it is possible that at any time there might be blocked jobs from more than one station waiting to enter station $j$. The maximum number of blocked jobs will be equal to the number of stations' servers which are directly connected to station $j$. We assume that these blocked jobs enter station $j$ on a first-blocked-first-enter basis.

With these assumptions, we obtain the queueing network model which is classified as transfer blocking case. Let $k^*$ denote the state of the network without considering the station capacities.

$$k^* = (k_1^*, k_2^*, \ldots, k_N^*)$$

where $k_i^*$ is the number of jobs in the $i$th station.

There must be at least $(K - m_i)$ waiting places in each queue ($m_i$ jobs can be in service) to ensure that all states are feasible. Since the capacity of the stations is finite, it is clear that all states $k^*$ cannot be feasible. The feasible states for blocking networks are obtained by realizing that the number $k_i^*$ of jobs in the $i$th station may not exceed the station capacity $M_i$ ($k_i^* \leq M_i$). Blocking events which occur in networks with finite station capacities have the effect of rendering states exceeding the station capacities infeasible.

The basic idea is that the infeasible states which violate station capacities are normalized. The states are adapted to the allowed capacities of the stations. The normalization of the states exceeding station capacities is executed by using the following formula. This indicates that if the number of jobs in a state of a station exceeds its capacity, $k_i^* > M_i$, then the job distribution is normalized until no violation of the capacity limits exists:

$$k_i^* := \begin{cases} M_j & \text{if } j = i \\ k_i^* + (k_i^* - M_i) \frac{e_i p_{i}}{e_i (1 - p_{i})} & \text{if } j \neq i \\ \end{cases}$$

for all $j = 1, 2, \ldots, N. \tag{3}$

where $e_i$ is the mean number of visits that a job makes to station $i$ and is computed by

$$e_i = \sum_{j=1}^{N} e_{j} p_{j} \quad \text{for } i = 1, \ldots, N. \tag{4}$$

The informal interpretation of the formula (3) is: if the capacity of a station is exceeded in a state, the number of jobs in that station is set equal to its capacity $M_j$ (first term) and distribute the remaining number of jobs in the predecessor stations according to the transition probabilities (second term).

Note that in case of networks with arbitrarily connected stations, the transition probabilities $p_{ij}$ in the normalization procedure, (3), cause the number of jobs $k_i$ in a state to have noninteger values. All nonfeasible states $k^*$ of the queueing network are normalized by (3). The job distribution in each state is adapted to the capacity limit of each station in the blocking network:

$$f(k^*) = (k)$$

where $k$ is the normalized state for the blocking network. The function $f$ transforms the nonfeasible state $(k^*)$ to the feasible state $(k)$.

By the normalization procedure, an equivalent state space structure is obtained for the blocking network. The only difference between the two state space structures is that the jobs are distributed according to the station capacities in the state space of the blocking network.

For a closed queueing network model with transfer blocking, the equilibrium probability distribution for the feasible states is computed by

$$p(k) = \sum_{k^* \text{ where } f(k^*) = k} p^*(k^*). \tag{6}$$

$p^*(k^*)$ represents the equilibrium state probability distribution of the network without station capacity restrictions computed by the well-known Gordon/Newell theorem [10], [35], [40].

$$p^*(k_1^*, k_2^*, \ldots, k_N^*) = \frac{1}{G(K)} \prod_{i=1}^{N} \left[ \frac{e_i}{\mu_i \beta_i(k_i^*)} \right]^{k_i^*} \tag{7}$$

$G(K)$ represents the normalization constant which can be calculated by any product form network algorithm, e.g., the convolution algorithm [33], [35] or LBANC technique [35].

The function $\beta_i(k_i^*)$ is defined by

$$\beta_i(k_i^*) = \begin{cases} k_i^*! & \text{if } k_i^* \leq m_i \\ m_i! \left( \frac{m_i}{m_i - n^*} - m_i \right) & \text{if } k_i^* > m_i \end{cases} \tag{8}$$
Performance measures such as the mean number of jobs $E_i$ at the $i$th station are computed by

$$E_i(K) = \sum_{n=1}^{M_i} np_i(n).$$  \hfill (9)

The marginal probability of $n$ jobs in the $i$th station, $p_i(n)$, is computed from the equilibrium state probabilities $p(k)$, (7):

$$p_i(n) = \sum_{k=1}^{K_i} p(k).$$  \hfill (10)

As mentioned before, the normalized state for blocking networks with arbitrarily connected stations may be noninteger values for the number of jobs. Thus, for networks with arbitrarily connected stations, the mean number of jobs is computed by

$$\bar{E}_i(K) = \sum_{k=1}^{K_i} f_i(k^*)p(k) \text{ for } i = 1, \ldots, N.$$  \hfill (11)

where the function $f_i(k^*)$ is the $i$th component of the function $f(k^*)$.

Even though all performance measures can be computed from the equilibrium state probabilities using the according formulas, we discovered this holds only for queueing networks with infinite station capacities. For example, in order to obtain the throughput in blocking queueing networks, we could not use the following well-known throughput formula [35]:

$$\lambda_i(K) = \sum_{n=1}^{K_i} \mu_i(n)p_i(n)$$  \hfill (12)

where the marginal probabilities are obtained by (10) and the load dependent service rates are given by

$$\mu_i(n) = \begin{cases} n\mu_i & \text{if } n \leq m_i \\ m_i\mu_i & \text{if } n > m_i \end{cases}$$  \hfill (13)

We were not able to obtain other performance measures by the equilibrium state probabilities. Hence, we looked for other ways and found the following concept for the computation of the throughput values in blocking queueing networks, as was also shown in [3] for single server cases.

**III. Throughput Analysis in Blocking Queueing Networks**

In [1], we have shown that the state space of two-station networks with blocking is isomorphic to the state space of two-station nonblocking networks with appropriate total number of jobs. Based on this concept, we obtained exact results for two-station networks with blocking. In the case of networks with more than two stations, we have realized that the state space of the blocking network cannot be transformed bijectively into the state space of the nonblocking network. However, the number of states in both networks may be equal or approximately equal. This would imply that both networks have approximately the same stochastic structure and the throughputs of both networks are approximately equal.

The following steps are executed in order to compute the throughput values in queueing networks with blocking.

1) Determine the number of states in the blocking queueing network.

2) Determine the total number of jobs $\bar{K}$ in the nonblocking queueing network.

3) Analyze the nonblocking queueing network with $\bar{K}$ jobs using MVA [34] or LBANC technique [35] and obtain the throughput value which is approximately equal to the throughput value of the blocking network with $K$ jobs.

**A. Determine the Number of States in Blocking Queueing Networks**

As mentioned before, in blocking networks, each station has a capacity limit, which indicates that a certain number of states can be feasible. The feasible states for blocking networks are obtained by realizing that the number $k_i$ of jobs in the $i$th station may not exceed its capacity $M_i$, $k_i \leq M_i$. Blocking events which occur in networks with finite station capacities must also be taken into account. Therefore, the $m_i$ neighbors of the feasible states are included, representing the blocking states for the stations. As many servers a station possesses as many neighbors of the feasible states are included to the state space. Whenever a transition occurs from one state to another state in which the capacity limit of a station would be violated, we assume that the transition causes a blocking action in the network and that the state entered is a blocking state. In reality, the job still resides in the source station. All the other states are infeasible and are cancelled.

In this way, we obtain a substate space for the blocking network, with $Z'(K)$ number of feasible and blocking states. For the computation of $Z'(K)$ we have developed an efficient convolution algorithm [3] which is extended here to multiple server cases.

The number of the feasible and blocking states is obtained from the last element $Z'(K)$ of the following vector result $Z'$:

$$Z' = Z_1 \otimes Z_2 \otimes \cdots \otimes Z_N$$  \hfill (14)

where $\otimes$ is a convolution operation which is explained as follows.

Let $A$, $B$, and $C$ be three $(N+1)$-dimensional vectors. The convolution operation $\otimes$ gives the following result:

$$C = A \otimes B$$

with

$$c(i) = \sum_{i=0}^{K} a(i) \cdot b(i-1) \text{ for } i = 0, 1, \ldots, K.$$  \hfill (15)

$Z_i$, for $i = 1, 2, \ldots, N$ is a $(K+1)$-dimensional vector given by

$$Z_i = \begin{bmatrix} z_i(0) \\ z_i(1) \\ \vdots \\ z_i(K) \end{bmatrix}$$
IZ: PRODUCT FORM APPROXIMATIONS FOR QUEUEING NETWORKS

with the binary function as elements

\[
z_i(k) = \begin{cases} 
1 & \text{if } k = 0, 1, 2, \cdots, \left( M_i + \sum_{j=1}^{N} m_j \right) \\
0 & \text{otherwise.}
\end{cases}
\]

Note that if the stations have no capacity limitations, then \( Z'(K) \) is equal to \( Z \), the known binomial coefficient formula:

\[
Z = \binom{N + K - 1}{N - 1}.
\]

This coefficient indicates the number of possible ways to distribute \( K \) jobs into \( N \) stations and provides the number of states \( Z \) in a closed queueing network without station capacity restrictions.

3. Determine the Total Number of Jobs \( \bar{R} \) in the Nonblocking Network

We have shown that the number of states \( Z'(K) \) (feasible and blocking states) in the blocking network is obtained by (14). The goal is to find a total number of jobs \( \bar{R} \) in a network with infinite station capacities having \( Z \) states such that \( Z \) will be approximately equal to \( Z'(K) \).

The total number of jobs \( \bar{R} \) in the nonblocking queueing network is determined by the following binomial coefficient formula:

\[
2 = \binom{N + K - 1}{N - 1}.
\]

If we want to make \( Z = Z'(K) \), it is possible that noninteger values for \( \bar{R} \) will result from (16). Since the number of jobs in networks is an integer value, we assume \( \bar{R} \) as the integer value which provides the nearest value to the number of states as \( Z'(K) \) computed by (14). Note that if a value for \( Z'(K) \) is obtained by (14) which is in the middle of two computed \( Z \) values, we choose that value for total number of jobs \( \bar{R} \) which provides the higher \( Z \) value.

C. Determine the Throughput of the Nonblocking Network

By analyzing the nonblocking network with \( \bar{R} \) total number of jobs using a product form algorithm such as mean value analysis [34], we obtain the total throughput \( \lambda_{NB}(\bar{R}) \). As mentioned before, since the number of states in both networks is approximately equal, it implies that the Markov processes describing the evolution of both networks have approximately the same structure. Consequently, the throughput of the nonblocking network \( \lambda_{NB}(\bar{R}) \) is approximately equal to the throughput of the blocking network \( \lambda_{B}(K) \).

\[
\lambda_{B}(K) = \lambda_{NB}(\bar{R}).
\]

The throughput \( \lambda_{i} \) and the utilization \( \rho_{i} \) of each station (for \( i = 1, \ldots, N \)) are computed by

\[
\lambda_{i}(K) = e_{i} \lambda_{i}(K) \quad \rho_{i}(K) = \frac{\lambda_{i}(K)}{m_{i} \mu_{i}}.
\]

The total response time \( \bar{t}_{C} \) (= cycle time = turnaround time) of jobs in the network is obtained by

\[
\bar{t}_{C}(K) = \frac{K}{\lambda_{0}(K)}.
\]

The mean residence time \( \bar{t}_{i} \) is determined using Little's law

\[
\bar{t}_{i}(K) = \frac{\bar{r}_{i}(K)}{\lambda_{0}(K)}.
\]

In the following numerical examples, we outline the general flow of both techniques.

IV. NUMERICAL EXAMPLES

A. Cyclic Network

We examine a queueing network with \( N = 3 \) serially switched stations and \( K = 6 \) jobs shown in Fig. 1. The stations have finite capacities as \( M_1 = 3, M_2 = 2, M_3 = 2 \) and each has \( m_1 = 2 \) servers (for \( i = 1, 2, 3 \)). The service times are exponentially distributed with mean values \( 1/\mu_1 = 1 \) s, \( 1/\mu_2 = 1.5 \) s, and \( 1/\mu_3 = 2 \) s.

The state space for this network has the structure shown in Fig. 2.

In Fig. 2, \( a = 2 \mu_1, b = 2 \mu_2, c = 2 \mu_3, a_1 = \mu_1, b_1 = \mu_2, \) and \( c_1 = \mu_3 \) denote the transition rates between the states.

As can be seen from the state space, Fig. 2, only the states \((3, 2, 1), (3, 1, 2), \) and \((2, 2, 2)\) are feasible. All other states exceed the capacity limits of the stations and hence they must be normalized. For example, the nonfeasible state \((6, 0, 0)\) is normalized as follows: the capacity of the first station is violated so we set the number of jobs equal to its capacity.

\[
k_1 = 3.
\]

The remaining three jobs are distributed to other stations:

\[
k_2 = k_2 + (k_1 - M_1)p_{21} = 0 \quad k_3 = k_3 + (k_1 - M_1)p_{31} = 3.
\]

By normalization we arrived at the state \((3, 0, 3)\) where the capacity of the third station is violated. So we set the number of jobs equal to the capacity of the third station.

\[
k_3 = 2.
\]

The remaining one job is put to the predecessor station 2:

\[
k_2 = k_2 + (k_2 - M_2)p_{23} = 1 \quad k_1 = k_1 + (k_2 - M_2)p_{12} = 3.
\]
As a result, the state $(6, 0, 0)$ becomes assigned the normalized job distribution $(3, 1, 2)$; see (3):

\[ f(6, 0, 0) = (3, 1, 2). \]

In another nonfeasible state $(0, 3, 3)$, the capacity of the second and third station is violated. The number of jobs is set equal to the, for example, third station's capacity.

\[ k_3 = 2. \]

The remaining job is put to the second station,

\[ k_1 = k_1 + (k_3 - M_3) p_{13} = 0 \]
\[ k_2 = k_2 + (k_3 - M_3) p_{23} = 4 \]

which causes the second station to violate its capacity.

A further normalization step provides that the state $(0, 3, 3)$ becomes assigned the normalized job distribution $(2, 2, 2)$; see (3):

\[ f(0, 3, 3) = (2, 2, 2). \]

After normalization of all nonfeasible states, we obtain the state space in Fig. 3.

First ignore the station capacities and compute the equilibrium state probabilities by (7):

\[
\begin{align*}
p*(6,0,0) &= 0.001 \\
p*(5,1,0) &= 0.003 \\
p*(4,2,0) &= 0.004 \\
p*(3,3,0) &= 0.007 \\
p*(2,4,0) &= 0.01 \\
p*(1,5,0) &= 0.007 \\
p*(0,6,0) &= 0.01 \\
p*(5,0,1) &= 0.004 \\
p*(4,1,1) &= 0.012 \\
p*(3,2,1) &= 0.018 \\
p*(2,3,1) &= 0.028 \\
p*(1,4,1) &= 0.042 \\
p*(0,5,1) &= 0.031 \\
p*(4,0,2) &= 0.0083 \\
p*(3,1,2) &= 0.025 \\
p*(3,0,3) &= 0.016 \\
p*(2,1,3) &= 0.031 \\
p*(1,2,3) &= 0.075 \\
p*(1,0,5) &= 0.066 \\
p*(0,2,4) &= 0.075 \\
p*(0,1,5) &= 0.099 \\
p*(0,0,6) &= 0.066.
\end{align*}
\]

The equilibrium state probabilities for the feasible states of the blocking network are computed by (6):

\[
\begin{align*}
p*(3,1,2) &= p*(6,0,0) + p*(5,1,0) + p*(5,0,1) + p*(4,1,1) \\
&+ p*(4,0,2) + p*(3,1,2) + p*(3,0,3) = 0.0704 \\
p*(3,2,1) &= p*(4,2,0) + p*(3,3,0) + p*(2,4,0) \\
&+ p*(1,5,0) + p*(0,6,0) + p*(3,2,1) \\
&+ p*(2,3,1) + p*(1,4,1) + p*(0,5,1) = 0.1702 \\
p*(2,2,2) &= p*(2,2,2) + p*(1,3,2) + p*(0,4,2) + p*(2,1,3) \\
&+ p*(1,2,3) + p*(0,3,3) + p*(2,0,4) \\
&+ p*(1,1,4) + p*(0,2,4) + p*(1,0,5) \\
&+ p*(0,1,5) + p*(0,0,6) = 0.7581.
\end{align*}
\]

The mean number of jobs $k_i$ in each station is computed from (9):

\[ k_1 = 2.238 \quad k_2 = 1.927 \quad k_3 = 1.827. \]

The throughput of the blocking network is determined by the throughput analysis method. First the number of states in
Fig. 3. Normalized state space.

Fig. 4. State space of the blocking network.

The number of states in this model is obtained from the last element $Z'(6) = 21$ of the vector result $Z'$. By drawing the state space in Fig. 4, the feasible states and their two immediate neighbors (two neighbors are required $m_i = 2$ as servers) representing blocking states, provide the same result $Z'(6) = 21$ as the algorithm, (14). The innermost dotted lines in Fig. 4 contain the feasible states where the outermost dotted lines contain their neighbors as blocking states. The states where capacity violations occur are cancelled.

The total number of jobs $K$ in the nonblocking network is obtained by (16):

$$Z = \binom{K+2}{2}.$$  

By varying the number of jobs $K$, we obtain different number of states in the nonblocking network as shown in Table I. Since the nonblocking network with $K = 5$ jobs possesses $Z = 21$ states which is equal to the number of states $Z'(6) = 21$ of the blocking network, we assume that the nonblocking network has $K = 5$ jobs. The state space for $K = 5$ is shown in Fig. 5.

Obviously, the state space structure (Fig. 5) is different from state space structure of the blocking network, Fig. 4. Hence, no unique transformation of the states is possible such that the results could be exact. However, the number of states...
TABLE II

<table>
<thead>
<tr>
<th>Station</th>
<th>$M_i$</th>
<th>$1/\mu_i$</th>
<th>$P_{11}$</th>
<th>$P_{12}$</th>
<th>$P_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.25</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

The mean number of visits $e_i$ is computed by (4)

\[ e_1 = 1 \quad e_2 = 1.955 \quad e_3 = 1.378. \]

We check the deadlock-free property, (2). In the cycle between stations 1 and 2, the total number of jobs is less than the total capacity of both networks, i.e.,

\[ \{K=3\} < \left\{ \sum_{j=1}^{2} M_j = 5 \right\} . \]

In the cycle between stations 1 and 3, condition (2) also holds:

\[ \{K=3\} < \left\{ \sum_{j=3}^{3} M_j = 4 \right\} . \]

also in the cycle between stations 2 and 3, condition (2) holds:

\[ \{K=3\} < \left\{ \sum_{j=2}^{3} M_j = 4 \right\} . \]

and finally in the cycle 2 and 2, condition (2) is again satisfied, i.e.,

\[ \{K=3\} < \{M_2 = 4\} . \]

The state space of the given network without considering the station capacities is shown in Fig. 7.

In Fig. 7, $a = \mu_1 p_{11}; b = \mu_2 p_{11}; c = \mu_2 p_{12}; d = \mu_3 p_{32}; e = \mu_3 p_{13}; f = \mu_3 p_{21}; g = \mu_3 p_{23}$ denote the transition rates between the states. It is obvious that the states $(3, 0, 0), (2, 1, 0)$, and $(2, 0, 1)$ are nonfeasible. These states exceed the first station's capacity and must be normalized using (3).

In the state, $(3, 0, 0)$, the number of jobs in the first station exceeds the capacity ($M_1 = 1$), so it is set equal to its capacity:

\[ k_1 = 1 \]

the remaining jobs are distributed as follows:

\[ k_2 = k_2 + (k_1 - M_1) \frac{e_2 p_{21}}{e_1 (1 - p_{11})} = 1.173 \]

\[ k_3 = k_3 + (k_1 - M_1) \frac{e_2 p_{31}}{e_1 (1 - p_{11})} = 0.8268. \]

Since none of the station capacities are violated, the normalization procedure yields

\[ f(3,0,0) = (1.173, 0.8268). \]

This is the situation that is mentioned before. The transition probabilities cause noninteger values for the job distributions in a normalized state.

Note that NOCAP in Table II denotes that the network is analyzed by ignoring the finiteness of the station capacities (NOCAPacity). The deviations $\delta$ are computed using the following relative deviation formula:

\[ \delta = \left\{ \frac{\text{Simulation Value} - \text{Analytical Value}}{\text{Simulation Value}} \right\} \times 100. \]
The equilibrium probabilities for the feasible states of the blocking network are computed by (6):

\[
\begin{align*}
p^*_{(1,1,1)} &= p^*_{(1,1,1)} = 0.0694 \\
p^*_{(1,2,0)} &= p^*_{(1,2,0)} = 0.0615 \\
p^*_{(0,2,1)} &= p^*_{(0,2,1)} = 0.1695 \\
p^*_{(3,0,0)} &= 0.0103 \\
p^*_{(1,1,0)} &= 0.00694 \\
p^*_{(2,1,0)} &= 0.0251 \\
p^*_{(0,0,3)} &= 0.2157.
\end{align*}
\]
From (11) we calculate the mean number of jobs \( k_i \) in each station, e.g.,

\[
k_1 = f_1(3,0,0)p(3,0,0) + f_1(2,1,0)p(2,1,0)
+ f_1(1,2,0)p(1,2,0) + f_1(2,0,1)p(2,0,1)
+ f_1(1,1,1)p(1,1,1) + f_1(1,0,2)p(1,0,2) = 0.273.
\]

Similarly we calculate the mean number of jobs in stations 2 and 3.

The total throughput, (17), and the throughput of each station, (18), and the mean response time \( t_r \), (20), are computed, listed, and compared with exact results in Table III.

V. EVALUATION

We have introduced two different techniques for the computation of performance measures in blocking networks with multiple servers. Both techniques have been implemented on a VAX 11/780 system. For the validation of both techniques, we have executed 200 network examples with different structures. Half of the examples were networks with arbitrarily connected stations. Each network model is also analyzed by varying the number of jobs. The number of jobs is varied from 5 to 100, the number of stations from 3 to 8, and the number of servers from 1 to 8. All examples have been simulated using the RESQ package [35].

Throughout the tests, we could not find any network model for which our approximations demonstrated instabilities. In all examples we have observed the same behavior. The fewer the jobs in the queueing network with finite station capacities, the less the chance for blocking. After a certain number of jobs in the network is reached, blocking events start to occur. As a result, the throughput does not increase with the number of jobs in the network. As the number of jobs in the network approaches the total capacity of the network, the more blocking events may occur. This has the effect of reducing the state space of the blocking network, which can be seen in Fig. 9. The solid line shows the number of states for a blocking queueing network consisting of three stations with station capacities \( M_1 = 12, M_2 = 10, \) and \( M_3 = 14, \) the number of servers \( m_1 = 4, m_2 = 2, \) and \( m_3 = 5 \) and the dashed line shows the number of states of the same network without station capacity limitations for different numbers of jobs. By dotted line, we show that the number of states in the blocking network is significantly reduced when the stations have single servers. It is obvious that the station capacities have the effect of decreasing the number of states as the number of jobs approaches the total capacity of the network.

In our test cases, most of the deviations for the throughput values are below 2 percent, implying that our throughput technique provides very accurate results for throughput and utilization. As can easily be seen in the chart in Fig. 10, the throughput values of the 65 examples in case of tandem networks (shaded portions) and of 43 in case of nontandem networks lie in the range of 0-1 percent deviation.

The major advantage of this technique is that the throughput values are obtained from a product form network which can easily be analyzed by mean value analysis or by any other product form network algorithm. Since mean value analysis has the advantage that it is extremely fast, it follows that we can obtain results for even a large queueing network with blocking in a very short time.

For the computation of the mean number of jobs \( k_i \), we need the equilibrium state probabilities which are given by (6). The method has shown a very stable behavior in the computation of the \( k_i \) values. The vast majority of the deviations were in the range of 1-5 percent as shown in histogram 2 in Fig. 11.

Compared to the throughput values, the deviations for the mean number of jobs are higher which can be observed in both histograms 1 and 2. Extremely small numbers cause deviations of 35 percent which makes 6 percent of the cases in histogram 2. As generally known, small numbers can provide high deviations even though the numerical results are not too different. In the histogram 3, in Fig. 12, we show the
normalization occurs in case of the nonfeasible state (0, 3, 0). The capacity of the second station is exceeded. So we apply (3) and obtain

\[ k_3 = 1 \]
\[ k_1 = 0.766 \]
\[ k_2 = 1.234. \]

Here we see that the normalization procedure, (3), resets the number of jobs in the second station to its capacity. However, some of the remaining jobs cause the capacity of the third station to be exceeded. In this case, we set the number of jobs in the third station back to its capacity and obtain

\[ k_3 = 1 \]
\[ k_1 = 0.834 \]
\[ k_2 = 1.166. \]

This time the capacity of the second station is violated again. The normalization causes both stations to swap jobs back and forth to each other. However, the remaining number of jobs is shuffled to the first station in each new normalization step. This is shown in Table IV.

As it can easily be seen in Table IV, the state values are tending towards (1, 1, 1). Note that in the implementation of this algorithm, an epsilon value of (\( \epsilon = 10^{-3} \)) prevents long run times for the normalization in such situations where two or more stations effect each other. If, after a normalization, the number of jobs exceeding the capacity less than the \( \epsilon \) value, then the normalization is assumed to be done. In the example above, Table IV, the value for \( k_3 \) violates the capacity by exceeding it by only 0.044. There is no reason to continue the normalization procedure until the values (1, 1, 1) are reached. Similar situations also occur for states (0, 1, 2), (0, 0, 3), and (0, 2, 1), which are normalized to the state (1, 1, 1).

Suri and Diehl [38] also consider closed queueing networks with Type 1 (transfer) blocking. They assume that the first station must have a capacity greater than the number of jobs in the network. They apply Norton's theorem [27], [35], [40] and reduce each two-station pair to a single station with a variable size queue capacity that is easy to analyze. An approximation algorithm is derived for the total throughput of the network. In the following we compare our throughput results to those of Suri and Diehl [38] and plot the throughput results for three blocking queueing networks with various number of jobs. Each graph contains three computations of total throughputs for various number of jobs: a) our algorithm, b) exact or simulation, c) Suri/Diehl results.
Example 1:

<table>
<thead>
<tr>
<th>Station</th>
<th>$M_i$ ((1/p_i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Fig. 13. Total throughput dependent on the number of jobs.

Example 2:

<table>
<thead>
<tr>
<th>Station</th>
<th>$M_i$</th>
<th>$1/p_i$</th>
<th>Exact</th>
<th>Our Approx.</th>
<th>$t$</th>
<th>Suri/Diehl</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>0.0099</td>
<td>0.00997</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>10</td>
<td>0.0081</td>
<td>0.0081</td>
<td>0</td>
<td>0.0081</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
<td>0.0782</td>
<td>0.0782</td>
<td>0</td>
<td>0.0782</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>10</td>
<td>0.0483</td>
<td>0.0483</td>
<td>0</td>
<td>0.0483</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>5</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0</td>
<td>0.0100</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>0.0867</td>
<td>0.0867</td>
<td>7.7</td>
<td>0.0867</td>
<td>7.7</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>0.0649</td>
<td>0.0649</td>
<td>0</td>
<td>0.0649</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>10</td>
<td>0.0782</td>
<td>0.0782</td>
<td>0</td>
<td>0.0782</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>0.0472</td>
<td>0.0472</td>
<td>4.3</td>
<td>0.0472</td>
<td>4.3</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>5</td>
<td>0.0229</td>
<td>0.0229</td>
<td>0</td>
<td>0.0229</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.0867</td>
<td>0.0867</td>
<td>0</td>
<td>0.0867</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
<td>0.0861</td>
<td>0.0861</td>
<td>4.9</td>
<td>0.0861</td>
<td>4.9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.0861</td>
<td>0.0861</td>
<td>0</td>
<td>0.0861</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>50</td>
<td>0.0443</td>
<td>0.0443</td>
<td>4.6</td>
<td>0.0443</td>
<td>4.6</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>0.0459</td>
<td>0.0459</td>
<td>8.6</td>
<td>0.0459</td>
<td>8.6</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>10</td>
<td>0.0459</td>
<td>0.0459</td>
<td>0</td>
<td>0.0459</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
<td>0.0472</td>
<td>0.0472</td>
<td>6.4</td>
<td>0.0472</td>
<td>6.4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.0472</td>
<td>0.0472</td>
<td>1.4</td>
<td>0.0472</td>
<td>1.4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>50</td>
<td>0.0377</td>
<td>0.0377</td>
<td>2.8</td>
<td>0.0377</td>
<td>2.8</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>100</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>100</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>50</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>0.0099</td>
<td>0.0099</td>
<td>0</td>
<td>0.0099</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 14. Total throughput dependent on the number of jobs.

Example 3:

- $N = 10$ stations; $M_1 = \infty$; $M_i = 3$ for $i = 2, \ldots, 10$; $\mu_i = 1$ for $i = 1, \ldots, 10$.

Example 5:

<table>
<thead>
<tr>
<th>$1/p_1$</th>
<th>$1/p_2$</th>
<th>$1/p_3$</th>
<th>Exact</th>
<th>Our Approx.</th>
<th>$t$</th>
<th>Suri/Diehl</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
<td>0.0714</td>
<td>0.0094</td>
<td>2.8</td>
<td>0.0094</td>
<td>2.8</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.0613</td>
<td>0.0095</td>
<td>2.9</td>
<td>0.0095</td>
<td>2.9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>0.0420</td>
<td>0.0115</td>
<td>1.6</td>
<td>0.0115</td>
<td>1.6</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>10</td>
<td>0.0417</td>
<td>0.0423</td>
<td>1.3</td>
<td>0.0423</td>
<td>1.3</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>20</td>
<td>0.0330</td>
<td>0.0340</td>
<td>2.8</td>
<td>0.0340</td>
<td>2.8</td>
</tr>
</tbody>
</table>

As can easily be seen in the graphs and also in the tables, both techniques provide very accurate results. However, the method of Suri and Diehl has the disadvantage that it provides only the total throughput (and using Little's law the total response time) of the network. In other words, performance measures for individual stations cannot be obtained. Another restriction is the assumption that the first station must have an infinite capacity. Additional restriction of the Suri–Diehl algorithm is that it is applicable only to closed tandem networks with single servers where our algorithm, Section III, is also applicable to nontandem networks with multiple servers. Our study also showed that the method of Suri and Diehl needs long run time (even though it is short compared to simulation) because of the computation of the marginal probabilities which are then used as the variable buffer sizes for the flow-equivalent server. Another major drawback in Suri and Diehl's method as mentioned in Section I, is that the capacity of each downstream station must be smaller than the total number of jobs in the network. This fact can be observed in examples 1, 2, and 3 where the results for Suri and Diehl do not appear for all possible number of jobs in the network.

In conclusion, we point out that our approximate techniques can be considered as an efficient way to compute the performance measures of queueing networks with blocking. The approximations are stable and the results obtained are accurate. The run times are short in particular for the computation of the throughput values.

In the following tables, we give total throughput results for different network examples which are taken from Suri and Diehl [33].
**APPENDIX**

We give results for 10 blocking queueing networks with various number of jobs. We have a total of 25 test cases with different system input parameters. Each table contains three repetitions of results for various number of jobs: 1) our algorithm simulation, and 3) NOCAP. The third column contains the standard deviation of simulation results. $\delta_1$ shows the relative deviations between our results and the simulation. $\delta_2$ shows the deviations between NOCAP and the simulation. This column demonstrates the effects of finite station capacity on the performance of the network.

**Example 1. (Cyclic):**

<table>
<thead>
<tr>
<th>Station</th>
<th>M</th>
<th>(1/m)</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| Example 3. (Cyclic): |

<table>
<thead>
<tr>
<th>Station</th>
<th>M</th>
<th>(1/m)</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>1.2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1.4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>1.3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>1.3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>1.2</td>
<td>4</td>
</tr>
</tbody>
</table>

| a) K = 25 jobs |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>4.3238</td>
<td>4.9288</td>
<td>1.1</td>
<td>0.1</td>
<td>4.1810</td>
</tr>
<tr>
<td>$J_2$</td>
<td>6.3819</td>
<td>6.9996</td>
<td>2.8</td>
<td>1.7</td>
<td>5.3110</td>
</tr>
<tr>
<td>$J_3$</td>
<td>5.1553</td>
<td>5.810</td>
<td>4.4</td>
<td>4.3</td>
<td>4.6646</td>
</tr>
<tr>
<td>$J_4$</td>
<td>0.5708</td>
<td>0.5808</td>
<td>1.7</td>
<td>1.4</td>
<td>0.5488</td>
</tr>
<tr>
<td>$J_5$</td>
<td>4.1819</td>
<td>4.231</td>
<td>5.8</td>
<td>5.6</td>
<td>4.1816</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>2.4744</td>
<td>2.569</td>
<td>8.6</td>
<td>7.9</td>
<td>2.5017</td>
</tr>
</tbody>
</table>

| b) K = 50 jobs |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>15.8866</td>
<td>15.914</td>
<td>1.1</td>
<td>0.1</td>
<td>15.791</td>
</tr>
<tr>
<td>$J_2$</td>
<td>15.8995</td>
<td>15.911</td>
<td>1.2</td>
<td>0.1</td>
<td>15.791</td>
</tr>
<tr>
<td>$J_3$</td>
<td>0.7877</td>
<td>0.8288</td>
<td>4.1</td>
<td>3.4</td>
<td>0.7809</td>
</tr>
<tr>
<td>$J_4$</td>
<td>6.9779</td>
<td>7.282</td>
<td>4.4</td>
<td>3.5</td>
<td>6.9739</td>
</tr>
<tr>
<td>$J_5$</td>
<td>0.5158</td>
<td>0.5171</td>
<td>1.2</td>
<td>0.1</td>
<td>0.5193</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>3.7071</td>
<td>3.754</td>
<td>7.1</td>
<td>7.2</td>
<td>3.7019</td>
</tr>
</tbody>
</table>

| c) K = 60 jobs |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>15.8464</td>
<td>15.858</td>
<td>0.9</td>
<td>0.8</td>
<td>15.8092</td>
</tr>
<tr>
<td>$J_2$</td>
<td>15.8995</td>
<td>15.928</td>
<td>0.2</td>
<td>0.2</td>
<td>15.9069</td>
</tr>
<tr>
<td>$J_3$</td>
<td>6.9708</td>
<td>7.176</td>
<td>6.0</td>
<td>4.9</td>
<td>6.9077</td>
</tr>
<tr>
<td>$J_4$</td>
<td>15.4043</td>
<td>15.738</td>
<td>3.0</td>
<td>2.4</td>
<td>15.0008</td>
</tr>
<tr>
<td>$J_5$</td>
<td>15.3946</td>
<td>15.908</td>
<td>0.2</td>
<td>0.2</td>
<td>15.0024</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>2.3727</td>
<td>2.171</td>
<td>0.4</td>
<td>0.3</td>
<td>2.1511</td>
</tr>
</tbody>
</table>

**Example 2. (Cyclic):**

<table>
<thead>
<tr>
<th>Station</th>
<th>M</th>
<th>(1/m)</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| Example 4. (Cyclic): |

<table>
<thead>
<tr>
<th>Station</th>
<th>M</th>
<th>(1/m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>1.5</td>
</tr>
</tbody>
</table>

| a) K = 33 jobs |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>12</td>
<td>11.81</td>
<td>0.5</td>
<td>0.4</td>
<td>12.4146</td>
</tr>
<tr>
<td>$J_2$</td>
<td>7.009</td>
<td>7.321</td>
<td>0.6</td>
<td>0.4</td>
<td>7.0079</td>
</tr>
<tr>
<td>$J_3$</td>
<td>13.997</td>
<td>13.376</td>
<td>0.9</td>
<td>0.8</td>
<td>13.3514</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>1.5</td>
<td>1.5</td>
<td>0</td>
<td>0</td>
<td>1.5</td>
</tr>
</tbody>
</table>

| b) $m_1 = 2; m_2 = 5; m_3 = 5 |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>12</td>
<td>11.81</td>
<td>0.5</td>
<td>0.4</td>
<td>12.4146</td>
</tr>
<tr>
<td>$J_2$</td>
<td>7</td>
<td>7.009</td>
<td>0.5</td>
<td>0.4</td>
<td>12.4146</td>
</tr>
<tr>
<td>$J_3$</td>
<td>14</td>
<td>14</td>
<td>0.3</td>
<td>0.3</td>
<td>1.5</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>1</td>
<td>0.995</td>
<td>0.1</td>
<td>0.1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

| b) K = 50 jobs |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>33</td>
<td>34.97</td>
<td>0.07</td>
<td>0.1</td>
<td>34.829</td>
</tr>
<tr>
<td>$J_2$</td>
<td>15</td>
<td>13.49</td>
<td>0.3</td>
<td>0.3</td>
<td>14.099</td>
</tr>
<tr>
<td>$J_3$</td>
<td>0.15</td>
<td>0.15</td>
<td>0</td>
<td>0</td>
<td>0.153</td>
</tr>
<tr>
<td>$J_4$</td>
<td>0.49</td>
<td>1</td>
<td>0.49</td>
<td>0</td>
<td>0.501</td>
</tr>
<tr>
<td>$J_5$</td>
<td>0.24</td>
<td>0.24</td>
<td>0.6</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>1.333</td>
<td>1.333</td>
<td>0</td>
<td>0</td>
<td>1.333</td>
</tr>
</tbody>
</table>

| c) $m_1 = 3; m_2 = 8; m_3 = 5 |

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Simul.</th>
<th>Std. Dev.</th>
<th>l (%)</th>
<th>NOCAP</th>
<th>l (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>11.84</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>11.175</td>
</tr>
<tr>
<td>$J_2$</td>
<td>7</td>
<td>7.05</td>
<td>0.3</td>
<td>0.7</td>
<td>11.175</td>
</tr>
<tr>
<td>$J_3$</td>
<td>14</td>
<td>14</td>
<td>0.2</td>
<td>0.4</td>
<td>11.175</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>1.48</td>
<td>1.48</td>
<td>0.4</td>
<td>0.4</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Example 5. (Noncyclic):
\[ K = 9 \text{ jobs}; \]

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{Station} & \textbf{Mi} & \textbf{1 (1/j)} & \textbf{E} & \textbf{NOCAP} & \textbf{F} & \textbf{P} \\
\hline
1 & 5 & 2.6 & 0 & 0.5 & 0 & 0.5 \\
2 & 5 & 1.3 & 0.7 & 0 & 0.5 \\
3 & 5 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
\end{table}

Example 6. (Noncyclic):

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{Station} & \textbf{Mi} & \textbf{1 (1/j)} & \textbf{E} & \textbf{F} & \textbf{P} & \textbf{R} \\
\hline
1 & 5 & 2.6 & 0 & 0.5 & 0 & 0.5 \\
2 & 5 & 1.3 & 0.7 & 0 & 0.5 \\
3 & 5 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
\end{table}

Example 7. (Noncyclic):

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{Station} & \textbf{Mj} & \textbf{1 (1/j)} & \textbf{E} & \textbf{F} & \textbf{P} & \textbf{R} \\
\hline
1 & 5 & 2.6 & 0 & 0.5 & 0 & 0.5 \\
2 & 5 & 1.3 & 0.7 & 0 & 0.5 \\
3 & 5 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
\end{table}

Example 8. (Noncyclic):

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{Station} & \textbf{Mi} & \textbf{1 (1/j)} & \textbf{E} & \textbf{F} & \textbf{P} & \textbf{R} \\
\hline
1 & 5 & 2.6 & 0 & 0.5 & 0 & 0.5 \\
2 & 5 & 1.3 & 0.7 & 0 & 0.5 \\
3 & 5 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
\end{table}

Example 9. (Noncyclic):

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{Station} & \textbf{Mi} & \textbf{1 (1/j)} & \textbf{E} & \textbf{F} & \textbf{P} & \textbf{R} \\
\hline
1 & 5 & 2.6 & 0 & 0.5 & 0 & 0.5 \\
2 & 5 & 1.3 & 0.7 & 0 & 0.5 \\
3 & 5 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
\end{table}
For the sake of brevity, I'll provide a summary of the key points from the text:

**REFERENCES**


---


---


---


---


---


---


---


---


---


---


---

F. Akylidz (M'85) was born in 1954 in Istanbul, Turkey. He received the Vordiplan, Diplom Informatiker, and Doctor of Engineering degrees in computer science from the University of Erlangen-Nuernberg, West Germany in 1978, 1981, and 1984, respectively.

From 1981 through 1985 he served as a Scientific Employee in the Informatik IV (Operating Systems) at the University of Erlangen-Nuernberg. During that time, he coauthored a text book titled Analysis of Computer Systems (in German) published by Teubner Verlag in Fall 1982. In January of 1985, he joined the faculty of the Department of Computer Science, Louisiana State University as an Assistant Professor. He was also a Visiting Professor in the Department of Computer Science at the University of Florida, Gainesville, in the summer of 1983 and in the Computer Science Department of the Universidad Tecnica de Federico Santa Maria in Valparaiso, Chile in the summer of 1986. In Fall 1987, he joined the faculty in the School of Information and Computer Science at Georgia Institute of Technology as an Assistant Professor. His research interests are performance evaluation, operating systems, and computer networks.

Dr. Akylidz is a member of the Association for Computing Machinery (SIGOPS and SIGMETRICS), GI (Gesellschaft fuer Informatik), and MMB (German Interest Group in Measurement, Modeling and Evaluation of Computer Systems).
APPLICATION OF NORTON'S THEOREM ON
QUEUEING NETWORKS WITH FINITE CAPACITIES

I. F. Akyildiz and J. Liebeherr


School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
U. S. A.

July 15, 1988
Application of Norton’s Theorem on Queueing Networks with Finite Capacities

I. F. Akyildiz
J. Liebeherr

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

ABSTRACT

The application of Norton’s theorem from electrical circuit theory on queueing networks with infinite capacities is well-known and very useful for cases where a station of the network should be analysed under different workload. In this work a method is developed which allows the application of Norton’s theorem on queueing networks with blocking. A station is arbitrarily selected and the subnetwork containing all remaining stations are replaced by a composite station with infinite capacity. The entire network is reduced to two-station network having the station of interest and the composite station. Although blocking causes interdependencies between stations in the network the selected station is totally isolated from the rest of the network by constructing phases in the server which reflect the blocking events. An algorithm is given to compute the parameters of the phases. Several examples are discussed to demonstrate the efficiency and generality of the technique. Comparisons with simulation results show that the proposed technique provides accurate results for throughput values.

Key Words: Performance Evaluation, Queueing Networks, Parametric Analysis, Blocking, Throughput

July 10, 1988
Application of Norton’s Theorem on Queueing Networks with Finite Capacities

I. F. Akyildiz
J. Liebeherr

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

1. Introduction

Queueing networks have experienced a dramatic increase in their importance regarding performance evaluation of computer systems and communication networks. When considering systems in which the stations have infinite capacities, numerous methods have been introduced in the past two decades. However, since in actual systems stations have a finite capacity, queueing networks with blocking should be used for performance analysis. Queueing networks with blocking have thus become an important research topic within performance evaluation during recent years. Several computational methods have been developed to analyze queueing networks with blocking. These are networks where the stations have finite capacities, hence blocking can occur if the station is full to its capacity. A job which wants to come to the full station must reside in the server of the source station until a place is available in the destination station. The interest in networks with blocking comes primarily from the realization that these models are useful in the study of the behavior of subsystems of computers and communication networks, in addition to detailed descriptions of several computer-related applications such as flexible manufacturing systems. In this work we consider the so-called transfer blocking mechanism in queueing networks. In this case, the blocking event occurs when a job completing service at station i cannot proceed to station j because station j is full. The job resides in station i’s server, which stops processing until station j releases a job. This type of blocking has been used to model systems such as production systems and disk I/O subsystems.

This work was supported in part by School of Information and Computer Science, ICS, of Georgia Institute of Technology and by the Air Force of the Scientific Research (AFOSR) under Grant AFOSR-R-88-0128.
Several investigators in recent years have published results on queueing networks with transfer blocking. Since we are investigating closed queueing networks with transfer blocking we discuss here the previous work for this type of networks. Akyildiz [3] studied two-station closed queueing networks with transfer blocking and multiple server stations. He showed that the equilibrium state probability distributions of such blocking systems are identical to those of a two-station closed queueing network without blocking. Akyildiz [5] also showed that the throughput of a blocking network with \( K \) total number of jobs is approximately equal to the throughput of a non-blocking network with an appropriate total number of jobs \( \hat{K} \). The well-known mean value analysis algorithm [23] is extended by Akyildiz [6] to single server queueing networks with blocking. The approximation is based on the modification of mean residence times due to the blocking events that occur in the network. Two algorithms for the computation of throughput values and the mean queue lengths in Markovian blocking queueing networks with multiple servers is given in [7] which is extended in [4] to networks with general service time distributions and FCFS scheduling disciplines.

Suri/Diehl [26] developed a method for approximate analysis of closed tandem queueing networks with transfer blocking. They approximate groups of two stations by a variable capacity station, defined as a superposition of fixed capacity stations. They start with the last two stations and successively reduce the network until two stations in tandem remain. The method is easy to implement and shows good accuracy but involves much computation. At each step all conditional probabilities have to be found, since they are used to construct the equivalent variable capacity station. The major disadvantage of their technique is that one of the stations must have an infinite capacity. Additionally, their method only gives the throughput of the entire network it does not give statistics for individual stations.

Dallery/Frein [12] introduce an iterative technique to obtain performance measures for the same network configuration as investigated by Suri/Diehl. Their throughput values are generally less accurate than the method of Suri/Diehl. However, they obtain values for the mean number of jobs which cannot be computed by the method of Suri/Diehl. Perros, Nilsson and Liu [21] give an algorithm for an arbitrarily connected network where some stations have finite capacity. They partition the set of stations in a so-called blocking subnetwork and a non-blocking subnetwork. The non-blocking subnetwork contain-
ing infinite capacity stations is replaced by a composite station using parametric analysis for infinite capacity networks. The reduced network is then analyzed numerically. However, if all stations of the network have finite capacity this method reduces itself to a numerical analysis method which, as generally known, is applicable on very small networks. Onvural/Perros [16] present an approximation for cyclic networks with blocking which calculates throughput values as a function of the number of jobs. They initially calculate throughput values for certain populations and then generate a function which fits the determined points. Equivalencies between closed networks with different blocking mechanisms are studied by Onvural/Perros [17] where they show if the number of jobs in a network with transfer blocking is one more the capacity of the station with the smallest capacity there is an exact product form solution.

Our work is mostly motivated by the studies of [8, 9, 18, 22]. Although these results apply only on open queueing networks with blocking, the concept of constructing phases in order to represent blocking events helped us to execute the parametric analysis of closed queueing networks with blocking. In recent years several other investigators have published results on queueing networks with blocking. A bibliography concerning queueing network models with blocking is given by Perros [20]. A recent workshop gives also a good overview about the area of queueing networks with blocking [19].

2. Model Assumptions

We consider closed queueing networks with \( N \) stations and \( K \) total jobs. The service time at station \( i \) is exponentially distributed with mean value \( 1/\mu_i \) (for \( i = 1, \ldots, N \)). The scheduling discipline at each station is assumed to be FCFS. Each station has a fixed finite capacity \( B_i \) where \( B_i = (\text{queue capacity} + 1), (\text{for } i = 1, 2, \ldots, N) \). Cases in which the stations can have infinite capacity are also allowed, \( B_i = \infty \), (for some \( i = 1, 2, \ldots, N \)). Any station whose capacity exceeds the total number of jobs in the network can be considered to have infinite capacity. A job which is serviced by the \( i \)-th station proceeds to the \( j \)-th station with probability \( p_{ij} \), (for \( i, j = 1, 2, \ldots, N \)), if the \( j \)-th station is not full. That is, if the number of jobs in the \( j \)-th station, \( k_j \), is less or equal to \( B_j \) for \( j = 1, 2, \ldots, N \). Otherwise, the job is blocked in the \( i \)-th station until a job in the \( j \)-th station has completed its servicing and a place becomes available. Furthermore it is valid that
which implies that the total number of jobs $K$ in the network may not exceed the total station capacity of the entire network.

One of the most important problems to realize regarding blocking queueing networks is that finite station capacities and blocking can introduce the problem of system deadlock. Deadlock may occur if a job which has finished its service at station $i$'s server wants join station $j$, whose capacity is full. That job is blocked in station $i$. Another job which has finished its service at $j$-th station now wants to proceed to the $i$-th station, whose capacity is also full. It blocks station $j$. Both jobs are waiting for each other. As a result, a deadlock situation arises. The following assumption states that a closed queueing network containing finite station capacities is deadlock free if and only if for each cycle $C$ in the network the following condition holds [1]:

$$K < \sum_{i \in C} B_i$$

Simply stated, the total number of jobs in the network must be smaller than the sum of station capacities in each cycle. Since tandem queueing networks have only one cycle, this condition, equation (2), corresponds to equation (1). Equation (1) is a sufficient condition for tandem networks to be deadlock free.

3. Norton's Theorem Application on Queueing Networks

The parametric analysis is based on an application of Norton's Theorem from electrical circuit theory to queueing networks. Chandy, Herzog and Woo [11] showed that Norton's Theorem provides an exact analysis for product form queueing networks [10]. We explain this concept by considering a closed queueing network model with $K$ jobs and $N = 12$ stations as shown in Figure 1.
For this queueing network model an equivalent network is constructed where a station \( \sigma \), here station 12, is arbitrarily selected. All other \((N - 1)\) stations are replaced by a single station, called the composite (flow-equivalent) station as shown in Figure 2.

Let \( \mu_z(k) \) be the composite mean service rate, where \( k \) is the number of jobs at this composite station. These composite mean service rates \( \mu_z(k) \), \( (\text{for } k=1,...,K) \), are determined by analyzing a modified version of the given network, in which the selected station \( \sigma \) has been shorted, i.e., the mean service time of station \( \sigma \) is set equal to zero, as shown in Figure 3.
Figure 3.

The throughput $\lambda(k)$ of the subnetwork, which we refer as $\Gamma$, are calculated for all jobs $k=1,2,\ldots,K$. These computed throughputs $\lambda(k)$ are set equal to the composite mean service rates $\mu_c(k)$. The solutions of the network consisting of the selected $\sigma$ and the composite station $c$ given in Figure 2 are identical to those of the original network model given in Figure 1.

The parametric analysis of Chandy, Hersog and Woo [11] is very interesting for cases in which only one station (e.g., a CPU) in a queueing network model is to be analyzed under various system workload. As mentioned before it provides exact results for product form queueing networks. As generally known queueing networks with blocking do not possess exact product form solution due to interdependencies between stations caused by finiteness of capacities. Consequently the application of Norton’s theorem on queueing networks with blocking will provide approximate results.
4. Norton's Theorem Application on Queueing Networks with Blocking

The application of Norton's theorem on queueing networks with blocking is executed in four steps.

4.1. Construction of the Subnetwork

4.2. Computation of Throughput of the Subnetwork

4.3. Construction of the Phases for the Selected Station

4.4. Analysis of the Two-Station Network

These steps will be explained in detail in the following sections.

4.1. Construction of the Subnetwork

We obtain the subnetwork \( r \) by shortening the selected station \( s \), i.e., setting its service time equal to zero as in the case of infinite capacity queueing networks. However, in case of finite capacity queueing networks we have to take the blocking events into account which occur on the paths between the selected station \( s \) and the stations of the subnetwork \( r \). Therefore we assume the capacity of stations in the subnetwork \( r \) to be infinite which are direct successors of station \( s \). However, if a station is a successor of \( s \) and receives arrivals from other stations within the subnetwork \( r \) its capacity remains unchanged. By this way the subnetwork \( r \) is converted to subnetwork \( r' \) which is explained formally as follows:

Let \( \Psi = \{ n_1, n_2, \ldots, n_N \} \) be the set of all stations in the originally given network and \( \Psi_i \) be the set of all stations which contains all successors of station \( i \), i.e.,

\[
(\forall i, j) \left[ (1 \leq i, j \leq N \& i \neq j \& p_{ij} > 0) \rightarrow (n_j \in \Psi_i) \right]
\]  

(3)

Subnetwork \( r \) is transformed into \( r' \) as follows:

\[
(\forall j, m) \left[ (1 \leq j, m \leq N \& j \neq n_j \in \Psi \& n_j \notin \Psi_m) \rightarrow (B_j := \infty) \right]
\]  

(4)

We give an example to illustrate this theory. The queueing model is given in Figure 4 where we assume that there are \( N = 7 \) stations, \( K = 12 \) jobs and the capacity of each station is selected as
\[ B_1 = 3, B_2 = 4, B_3 = 4, B_4 = 3, B_5 = \infty, B_6 = 3, B_7 = 7. \]

Figure 4.
We select station 1 to be analyzed under various workload. For that reason we shorten station 1, i.e.,
we set its service time equal to zero and obtain the subnetwork \( \Gamma \) given in Figure 5.

Figure 5.
In subnetwork \( \Gamma \) it can easily be seen that stations 2, 3 and 4 are direct successors of the selected
station \( \sigma = 1 \). Since station 2 receives also arrivals from station 5 it keeps its finite capacity while sta-
tions 3 and 4 receive arrivals only from \( \sigma \) it follows that their capacity are assumed to be infinite in \( \Gamma' \)
as given in Figure 6. Everything else in \( \Gamma' \) remains the same as in \( \Gamma \).
Figure 6.

For substituting $\Gamma'$ by a composite (flow-equivalent) station we need to compute throughput values $\lambda(k)$ of $\Gamma'$.

4.2. Throughput Analysis of the Subnetwork

In order to obtain the throughput values of the subnetwork $\Gamma'$ which is a blocking network itself we apply the technique by Akyildiz [5, 7]. The basic concept is that the state space of the blocking queueing network with $K$ total number of jobs is transformed into the state space of a non-blocking queueing network with $\hat{K}$ total number of jobs. The number of states in both networks should be approximately the same, if not identical. This would indicate that Markov processes describing the evolution of both networks over time have approximately the same structure. That, in turn, would guarantee that the throughputs of both networks are approximately equal.

The following steps are executed in order to compute the throughput values in queueing networks with blocking [5, 7].

i) Determine the number of states in the blocking queueing network.

ii) Determine the total number of jobs $\hat{K}$ in the non-blocking queueing network. Note that $\hat{K}$ may be a non-integer number.

iii) Analyse the non-blocking queueing network $NB$ with $\hat{K}$ jobs using the $\alpha$-MVA [13] which is applicable to queueing networks with non-integer number of jobs, and obtain the
throughput values which are approximately equal to the throughput value of the blocking network \( \Gamma' \) with \( K \) jobs.

\[
\lambda_\Gamma(K) \approx \lambda_{NB}(K)
\]  

(5)

As described in the next section the construction of the equivalent network is much more complex than in the case of infinite capacity queueing networks which will be explained in the next section.

4.3. Phase Construction of the Selected Station

So far we have analyzed the subnetwork \( \Gamma' \) and replaced all stations in \( \Gamma' \) by so-called composite (flow-equivalent) station \( e \). The load-dependent service rates \( \mu_e(k) \) of this composite station are set equal to the throughput values \( \lambda_\Gamma \) of the subnetwork \( \Gamma' \) which are computed as described in section 4.2. We assume that the composite station has infinite capacity.

By assuming the capacity of \( \sigma \)'s successor stations in \( \Gamma' \) to be infinite we neglected the blocking events at station \( \sigma \) which might have occurred due to some full stations in \( \Gamma' \). However, the blocking events must be considered for station \( \sigma \), i.e., a job cannot leave station \( \sigma \) due to some full stations in the subnetwork. Therefore, we modify the service mechanism of station \( \sigma \) such that all delays a job might undergo due to blocking events in the originally given network can be represented.

For each possible blocking delay at stations of \( \Gamma' \) we add a service phase to station \( \sigma \). The connection between the added phases and the original server of station \( \sigma \) are the same as the transitions between the stations in the originally given network. However, blocking delays may not only be caused by station \( \sigma \)'s immediate successors but also by stations which occur in a cycle, \( CYC = (n_\sigma, n_{\sigma+1}, n_{\sigma+2}, \ldots, n_\sigma) \). Therefore, those stations which are in a cycle \( CYC \) will be considered in the phase construction. However, if \( PATH = (n_{\sigma+1}, n_{\sigma+2}, \ldots, n_{\sigma+\epsilon}) \) is a path in \( CYC \) where \( n_{\sigma+1} \) is an immediate successor of station \( \sigma \) with \( \sum_{n_i \in PATH} B_i \geq K \) then the stations \( n_{\sigma+\epsilon}, n_{\sigma+\epsilon+1}, \ldots \) are not considered in the phase construction. In other words, if the sum of station capacities in \( PATH \) exceeds the total number of jobs then the last station of \( PATH \) and all its successors in \( CYC \) are not taken into account in the phase construction for station \( \sigma \).

In the transfer blocking case a blocked job cannot leave a station until a place is available in the
destination station [6]. Therefore, the blocking time of a job at a station is equal to the mean remaining service time of the particular destination station. Based on this argument the pseudo service time of each phase which represents the blocking events, is equal to the mean service time of the according station in the originally given network.

Since blocking events occur with certain probability in the network, the delay phases in station $\sigma$ are entered due to the according branching probabilities $a_{ij}$ which are computed by:

$$a_{ij} = p_{ij} \cdot P_j(B_j + 1)$$

where $i, j$ are stations which appear in the phase construction of station $\sigma$, $p_{ij}$ are transition probabilities of the originally given network and $P_j(B_j + 1)$ are blocking probabilities which are determined iteratively as explained in the next section.

Let us continue with the example given in Figure 4 and show how to construct the phases for station 1. Obviously, station 1 can be blocked by station 4 which, in turn, can be blocked by station 7. Station 7 may also cause blocking of station 1 since it appears on the cycle 1, 4, 7, 1 and $\{B_4 + B_7 < K\}$. It means that station 7 must be considered in the delay phase construction for station 1. Consequently the phases of station 1 contain the delays caused by stations 4 and 7. As shown in Figure 4 station 5 has an infinite capacity. Therefore, station 1 cannot be blocked by station 5. Moreover, since $\{B_2 + B_3 + B_7 > K\}$, it is not possible that stations 2 and 6 are blocked and station 7 is full. Consequently, the phase for station 7 does not need to be added after the phase for station 6. In Figure 7 we show the complete phase construction for station 1.

![Figure 7](image-url)
4.4. Analysis of the Two-Station Network

So far we reduced the entire network to two-station network which contains finite capacity station \( \sigma \) with complete constructed phases and the composite (flow-equivalent) infinite capacity station \( e \) representing all other stations. The two-station network for the example given in Figure 4 has the following structure:

![Two-station network diagram](image_url)

As pointed out in equation (6) the blocking probabilities \( P_i(B_f + 1) \) need to be determined for the analysis of the two-station network. These parameters \( P_i(B_f + 1) \) and the desired throughput values \( \lambda \) of this two-station network given in Figure 8 which are also the throughput values of the originally given network, are computed by an iterative way.

Initially we set all branching probabilities \( a_{ij} \) between service and delay phases of station \( \sigma \) to zero and eliminate all phases. Station \( \sigma \) has then its originally given structure, i.e., it has exponentially distributed service time with mean value \( 1/\mu_\sigma \) and finite capacity \( B_\sigma \) as shown in Figure 9.

![Station structure diagram](image_url)
This network is analyzed by the method given in [3] and the total throughput \( \lambda \) is computed. Using \( \lambda \) the throughput of each station \( j \) is determined by

\[
\lambda_j = \lambda \cdot e_j \quad \text{for} \quad j = 1, \ldots, N
\]

(7)

where \( e_j \) is the mean number of visits that a job makes to station \( j \) and is given by

\[
e_j = \sum_{i=1}^{N} e_i \cdot p_{ij} \quad \text{for} \quad j = 1, \ldots, N
\]

(8)

Now these computed throughput values \( \lambda_j \) are used as the arrival rates for \([ M /M /1 / B_j + 1 ]\) stations which are considered in the phase construction of station \( \sigma \). Here we assume that each station in the originally given network behaves approximately as a single server station with exponentially distributed service times and Poisson arrivals. Recall that a job in the transfer blocking protocol is already served, a destination is chosen and the job is blocked in the server. Logically this blocked job occupies the \((B_j + 1)\)st place in the queue of the \( j \)th destination station. The probability that the \((B_j + 1)\)th space in station \( j \) is occupied, provides the probability that one or more predecessors of station \( j \) are blocked in the originally given network.

Hence, using the well-known formula for steady state probabilities of \([ M /M /1 / B_j ]\) stations the values for the blocking probabilities \( P_j(B_j + 1) \) are computed [14]:

\[
P_j(B_j + 1) = \begin{cases} 
\hat{\beta}_j^{B_j+1} \cdot \frac{1 - \hat{\beta}_j}{1 - \hat{\beta}_j^{B_j+1}} & \text{if} \quad B_j < K \\
0 & \text{if} \quad B_j \geq K
\end{cases}
\]

(9)

with

\[
\hat{\beta}_j = \frac{\hat{\lambda}_j}{\mu_j} \quad \text{for} \quad j = 1, \ldots, N
\]

(10)

where \( \hat{\lambda}_j \) is the effective input rate to station \( j \) which can be expressed in terms of the arrival rate \( \lambda_j \) at station \( j \):

\[
\hat{\lambda}_j = \lambda_j \cdot \left[ 1 - P_j(B_j + 1) \right] \quad \text{for} \quad j = 1, \ldots, N
\]

(11)

Equations (8) and (10) are used as a fixpoint iteration for computation of \( P_j(B_j + 1) \) values. Note that the convergency of this fixpoint iteration was shown by Altiok [8] for open queueing networks with
blocking. The values for \( P_j(B_j + 1) \) are then used for determining the branching probabilities \( a_{ij} \) from equation (6).

Let us continue here to discuss the example given in Figure 4. Since stations 2,3,4,6 and 7 appear in the phase construction for station 1 we consider these as individual \([M/M/1/B_j + 1]\) stations:

![Diagram of the example network](image)

Figure 10.

Using the fixpoint iteration equations (8) and (10) we compute \( P_j(B_j + 1) \) values, which are then used for determining the branching probabilities \( a_{ij} \) of delay phases in equation (6).

\[
\begin{align*}
a_{12} &= p_{12} \cdot P_2(B_2 + 1) \\
ap_{23} &= p_{23} \cdot P_3(B_3 + 1) \\
ap_{13} &= p_{13} \cdot P_3(B_3 + 1) \\
ap_{14} &= p_{14} \cdot P_4(B_4 + 1) \\
ap_{37} &= p_{37} \cdot P_7(B_7 + 1) \\
ap_{47} &= p_{47} \cdot P_7(B_7 + 1)
\end{align*}
\]

The two-station network has now a complicated structure since the branching probabilities \( a_{ij} \) in \( \sigma \) are not zero as shown in Figure 8. In order to analyze this type of networks efficiently we reduce the serving and delay phases of station \( \sigma \) to a Coxian server with two phases. Marie [15] showed that this transformation provides good results for stations with general service time distribution if the squared coefficient of variation is greater than 0.5. Once all the parameters of the phase type distribution illustrated in Figure 8 are known we can construct a Coxian-2 representation by determining its first...
moment $1/\mu_{\sigma}$ and the coefficient of variation $\hat{\epsilon}_{\sigma}$ and then fitting a Coxian-2 distribution according to the following equations:

$$\bar{\mu}_{\sigma 1} = 2 \cdot \hat{\mu}_{\sigma}$$  \hspace{1cm} (11)
$$\bar{\mu}_{\sigma 2} = \hat{\epsilon}_{\sigma}^2 \cdot \mu_{\sigma}$$  \hspace{1cm} (12)
$$\hat{\epsilon}_{\sigma 1} = \frac{1}{2 \cdot \hat{\epsilon}_{\sigma}^2}$$  \hspace{1cm} (13)

This transformation leads to the following queueing network model:

This network can be analyzed by the numerical method [25] or by the load-dependent method of Marie [2]. Note that these techniques must be slightly modified because one station has finite capacity and blocking can occur in the network. Additionally, we have to consider that a job in station $\sigma$'s server can be in two phases. In Figure 12 we show the state space diagram for the network given in Figure 11. The state $(l; p, n)$ describes the situation where $l$ jobs are in station $\sigma$, the job being served in station $\sigma$ is in the $p$-th phase and $n$ jobs are in the composite station. Blocking states are labeled with a 's'.

Figure 11.

Figure 12.
A job after being served in the first phase of station \( \sigma \) can either proceed to the second phase (with probability \( a_{\sigma 1} \)), equation (13), or leave the station to join the composite station \( \zeta \). Recall that the composite station \( \zeta \) has a load-dependent service rate. For the example given in Figure 4 where \( B_1 = 3 \) and the total number of jobs is \( K = 12 \) we obtain the following state space diagram:

From this diagram global balance equations can easily be derived and steady state probabilities can be obtained using the numerical technique or the load-dependent method of Marie. From the steady state probabilities the throughput values are computed. As mentioned before these throughput values are then used as the individual arrival rates \( \lambda_j \) for \( [M/M/1/B_j + 1] \) stations and the values of \( P_j(B_j + 1) \) are computed iteratively from (8) and (10). The values for \( P_j(B_j + 1) \) are then used for determining the branching probabilities \( a_{\beta j} \), equation (6) in the multi-phase server of Figure 7. Here the scheme of the iteration can be recognized. We repeat the analysis between the two-station network given in Figure 11 and \([M/M/1/B_1 + 1]\) stations given in Figure 10, each time with modified branching probabilities \( a_{\beta j} \), until convergency is reached. The iteration terminates when the absolute value of the difference between two consecutive \( \lambda \) values is smaller than a threshold value \( \epsilon \) where \( \epsilon \) is selected as \( \epsilon = 10^{-4} \). Note also that no computation is necessary for branching probabilities of a station if the capacity of that station is infinite, i.e., the capacity is greater or equal to the total number of jobs.
5. Algorithm Summary

(1) Select an arbitrary station \( \sigma \), set its service time equal to zero \( (1/\mu_\sigma = 0) \) and obtain subnetwork \( \Gamma \).

(2) Transform the subnetwork \( \Gamma \) into \( \Gamma' \) according to expression (4).

(3) Analyze \( \Gamma' \) with the throughput algorithm [5, 7] for finite capacity queueing networks and obtain \( \lambda_{\Gamma'}(k) \) for \( k = 1, \ldots, K \). Construct the composite station \( c \) with infinite capacity and load-dependent service rate:

\[
\mu_c(k) := \lambda_{\Gamma'}(k) \quad \text{for} \quad k = 1, \ldots, K
\]

(4) Construct a multi-phase server for station \( \sigma \) according to section 4.3.

(5) Iterate for \( t = 1, 2, \ldots \)

(5.1) Initialize

\[
P_j(B_j + 1) := 0 \quad \text{for all} \quad j = 1, \ldots, N
\]

\[
\lambda^{(0)} := 0
\]

(5.2)

(a) Calculate the branching probabilities \( a_{ij} \) between the phases of station \( \sigma \) by (8). Obtain the mean value \( 1/\mu_\sigma \) and the coefficient of variation \( \delta_\sigma \) of the service time distribution of station \( \sigma \).

(b) Represent the multi-server phases of station \( \sigma \) by a Cox-distribution with two phases from equations (11, 12) and (13).

(5.3) Solve the two-station network containing Cox-two server and the composite station \( c \) and obtain throughput \( \lambda^{(t)} \).

(5.4) For each station \( j \) which has a server in the phase of station \( \sigma \) solve the fixpoint iteration from equations (8) and (10).

(5.5) Terminate if \( | \lambda^{(t)} - \lambda^{(t-1)} | < \varepsilon \). Otherwise assign

\[
\lambda^{(t-1)} := \lambda^{(t)}
\]

and go to step (5.2).
6. Evaluation

After the termination of the iteration we obtain the total throughput of the two-station network which is also the total throughput of the originally given network. The advantage of the parametric analysis can be recognized when we study the entire network by changing only the parameters of the selected station $\sigma$. In that case we need to compute throughput values of subnetwork $\Gamma'$ only once as described in section 4.2 which remain unchanged throughout the analysis. All we need to do is to construct the phases of station $\sigma$ in the two-station network as described in section 4.4. The required computational effort is small since the equivalent network consists of only two stations.

In this section we demonstrate the application of the parametric analysis for blocking networks by analyzing three models. The first model is a Central Server Model with four I/O - devices, the second model is a tandem configuration with five stations and the third model is a network with arbitrarily connected stations. For each example we illustrate the steps of the method and give numerical results. For checking the accuracy of our technique we compare our results with simulation values obtained by using RESQ package [24]. Relative errors $\delta\%$ are calculated by:

$$\delta\% = \left| \frac{\text{simulated value} - \text{analytical value}}{\text{simulated value}} \right| \cdot 100$$

We also give the number of necessary iteration steps in order to demonstrate the speed of convergency of the phase construction. We also compare our results with results obtained by other algorithms for finite capacity queueing networks such as the methods of Akyildiz [5], Suri/Diehl [26] and Dallery/Frein [12]. The techniques of Suri/Diehl as well as of Dallery/Frein are restricted to tandem configurations where the first station must have an infinite capacity. Additionally, Dallery and Frein assume that blocking occurs only at immediate successors of each station.
Example 1.

The central server model is given in Figure 14a for which we want to analyze the behavior of the CPU (station 1 in Figure 14a) under different workload.

Parameters of the network are:

\[
\begin{array}{c|cccccc}
 i & 1 & 2 & 3 & 4 & 5 \\
 \hline
 B_i & 8 & 4 & 7 & 3 & 3 \\
 1/\mu_i & 2 & 3 & 2 & 4 & 5 \\
\end{array}
\]

\[P_{12} = 0.2; P_{13} = 0.5; P_{14} = 0.2; P_{15} = 0.1; \quad p_{j1} = 1 \quad \text{for } j = 2, 3, 4, 5.\]

Subnetwork \(\Gamma'\) is obtained as described in section 4.1:
Note that in this case the subnetwork $\Gamma'$ contains only stations having infinite capacities. In other words, the subnetwork $\Gamma'$ has exact product form solution \([10,23]\). Hence we can apply an algorithm such as mean value analysis \([23]\) and obtain the following throughput values $\lambda(k)$.

![Figure 14c. Throughput values of the subnetwork $\Gamma'$](image)

In the phase construction for station 1 we have to consider the fact that station 1 can be blocked by all other stations 2, 3, 4 and 5. So we obtain the complete phases for station 1 as follows:

![Figure 14d.](image)

This station is connected to an infinite capacity station with a load-dependent service rate $\mu_i(k)$ equal to the throughput values $\lambda(k)$ of subnetwork $\Gamma'$. Then the iterative algorithm of section 4.4
applied. The results listed in Table 1 are compared with those obtained by [5]. Note that we assumed $K = 10$ as the maximum number of jobs in the following Table 1. Otherwise deadlock freedom, equation (2), would be violated.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Simulation</th>
<th>Parametric Analysis</th>
<th>$\delta%$</th>
<th>Akyildiz</th>
<th>$\delta%$</th>
<th>No. of Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.4402</td>
<td>0.4395</td>
<td>0.15</td>
<td>0.4396</td>
<td>0.13</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0.4840</td>
<td>0.4837</td>
<td>0.06</td>
<td>0.4855</td>
<td>0.3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0.4773</td>
<td>0.4782</td>
<td>0.16</td>
<td>0.4808</td>
<td>2.2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0.4837</td>
<td>0.4884</td>
<td>0.5</td>
<td>0.4894</td>
<td>1.1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0.4880</td>
<td>0.4905</td>
<td>0.5</td>
<td>0.4894</td>
<td>0.2</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>0.4884</td>
<td>0.4926</td>
<td>0.8</td>
<td>0.4943</td>
<td>1.2</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>0.4896</td>
<td>0.4926</td>
<td>0.6</td>
<td>0.4970</td>
<td>1.5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. Total Throughput of the Network

To show the advantage of our method we now vary the parameters of the selected station (station 1 in Figure 14a). We assign to station 1 the following values:

$$\frac{1}{\mu_1} = 5; \quad B_1 = 4$$

In the analysis of this network we do not need to consider the subnetwork $\Gamma'$ which was replaced by a composite station. We use the throughput results of $\Gamma'$ from Figure 14c and apply the iteration from section 4.4 and obtain the following values. In this case the maximum number of jobs allowed is $K = 6$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Simulation</th>
<th>Parametric Analysis</th>
<th>$\delta%$</th>
<th>Akyildiz</th>
<th>$\delta%$</th>
<th>No. of Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.1982</td>
<td>0.1982</td>
<td>0.0</td>
<td>0.1982</td>
<td>0.0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0.1997</td>
<td>0.1986</td>
<td>0.5</td>
<td>0.1996</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0.2005</td>
<td>0.108</td>
<td>0.3</td>
<td>0.1998</td>
<td>0.4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2. Total Throughput of the Network

Note that the structure of the multi-phase server of the selected station remains unchanged while the parameters of that station are modified.
Example 2.

In this case we consider a tandem configuration as given in Figure 15a.

![Figure 15a.](image)

Network parameters are:

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_i )</td>
<td>( \infty )</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( 1/\mu_i )</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Here we assume the capacity of the first station to be infinite. Therefore we will be able to compare our technique with the methods of Suri/Diehl [26] and Dallery/Frein [12] since their techniques are applicable only on this type of networks.

The subnetwork \( \Gamma' \) is shown in Figure 15b where we shorten station 1.

![Figure 15b.](image)
Analyzing the subnetwork $\Gamma'$ we obtain the following throughput values:

The phase construction for selected station 1 is given as follows:

Final results are listed in the following Table:

<table>
<thead>
<tr>
<th>K</th>
<th>Simulation</th>
<th>Par_An.</th>
<th>$\delta%$</th>
<th>AKYL</th>
<th>$\delta%$</th>
<th>SUDI</th>
<th>$\delta%$</th>
<th>DAFR</th>
<th>$\delta%$</th>
<th>No. of Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.367</td>
<td>0.3658</td>
<td>0.3</td>
<td>0.367</td>
<td>0.3</td>
<td>0.364</td>
<td>0.8</td>
<td>0.356</td>
<td>3.0</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0.349</td>
<td>0.383</td>
<td>0.1</td>
<td>0.390</td>
<td>0.3</td>
<td>0.381</td>
<td>2.1</td>
<td>0.382</td>
<td>1.8</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0.404</td>
<td>0.3975</td>
<td>1.6</td>
<td>0.407</td>
<td>0.7</td>
<td>0.392</td>
<td>3.0</td>
<td>0.400</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>0.413</td>
<td>0.4091</td>
<td>0.9</td>
<td>0.420</td>
<td>1.7</td>
<td>0.398</td>
<td>3.8</td>
<td>0.416</td>
<td>0.7</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>0.417</td>
<td>0.4144</td>
<td>0.6</td>
<td>0.420</td>
<td>0.7</td>
<td>0.400</td>
<td>4.2</td>
<td>0.426</td>
<td>2.1</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>0.419</td>
<td>0.4182</td>
<td>0.5</td>
<td>0.420</td>
<td>0.2</td>
<td>0.400</td>
<td>4.3</td>
<td>0.430</td>
<td>2.6</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>0.419</td>
<td>0.4178</td>
<td>0.2</td>
<td>0.420</td>
<td>0.2</td>
<td>0.401</td>
<td>4.3</td>
<td>0.430</td>
<td>2.6</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>0.419</td>
<td>0.4182</td>
<td>0.2</td>
<td>0.420</td>
<td>0.2</td>
<td>0.401</td>
<td>4.3</td>
<td>0.430</td>
<td>2.6</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>0.419</td>
<td>0.4182</td>
<td>0.1</td>
<td>0.430</td>
<td>2.6</td>
<td>0.401</td>
<td>4.3</td>
<td>0.430</td>
<td>2.6</td>
<td>3</td>
</tr>
</tbody>
</table>
Let us modify the parameters for the tandem network given in Figure 15a.

\[
\begin{array}{c|ccccc}
  i & 1 & 2 & 3 & 4 & 5 \\
  \lambda_i & 2 & 4 & 3 & 4 & 2 \\
  \mu_i & 1 & 2 & 1.5 & 1.8 & 1.6 \\
\end{array}
\]

The structure of subnetwork Γ' and the multi-phase server of the selected station 1 are the same as given in Figure 15b and Figure 15d, respectively. Since the parameters of stations in subnetwork Γ' are modified, the subnetwork Γ' must be analyzed again. We obtain the following throughput values for subnetwork Γ':

![Figure 15e.](image)

In Table 4 we compare our results with simulation and the method of Akyildiz. Since the capacity of station 1 is finite the methods of Suri/Diehl and Dallery/Frein cannot be applied.

<table>
<thead>
<tr>
<th>K</th>
<th>Simulation</th>
<th>Parametric Analysis</th>
<th>δ%</th>
<th>Akyildis</th>
<th>δ%</th>
<th>No. of Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.388</td>
<td>0.3631</td>
<td>6.4</td>
<td>0.3871</td>
<td>2.3</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>0.393</td>
<td>0.3775</td>
<td>3.9</td>
<td>0.3871</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>0.376</td>
<td>0.3787</td>
<td>2.3</td>
<td>0.3871</td>
<td>2.95</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>0.352</td>
<td>0.3787</td>
<td>7.5</td>
<td>0.3871</td>
<td>4.29</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4. Total Throughput of the Network
Example 3.

The network model has the following structure:

![Network Diagram](image)

**Figure 16a.**

Parameters for this example are chosen as follows:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_i$</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$1/\mu_i$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>0</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.4</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0</td>
</tr>
</tbody>
</table>

When we shorten station 1 in this network and construct subnetwork $\Gamma'$, we note that all stations in $\Gamma'$ have finite capacity. This is the case where each station receives arrivals from other stations within $\Gamma$. 
Analysis of $\Gamma'$ provides the following throughput values:

In this network blocking may occur only at immediate successor of each station. Otherwise deadlock freedom is not guaranteed [1]. Therefore we obtain a multi-phase server for station 1 as follows:
In Table 5 we show results for total throughput. Due to the deadlock freedom property from equation (2) the maximum number of jobs allowed in this network is $K = 8$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Simulation</th>
<th>Parametric Analysis</th>
<th>$\delta%$</th>
<th>Akyildis</th>
<th>$\delta%$</th>
<th>No. of Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.365</td>
<td>0.3734</td>
<td>2.3</td>
<td>0.375</td>
<td>2.7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>0.387</td>
<td>0.3983</td>
<td>2.8</td>
<td>0.398</td>
<td>2.8</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5. Total Throughput of the Network
7. Conclusions

In this work we introduced a method which shows the application of Norton's theorem on queueing networks with blocking. In section 2 we presented the model assumptions. The blocking mechanism we considered is the so-called transfer blocking. We also assumed that the networks investigated must be deadlock-free. In section 3 we reviewed briefly the application of Norton's theorem on queueing networks with infinite capacities. We then proposed an algorithm for queueing networks with finite capacities which enables us to profit from the advantages of applying Norton's theorem, i.e., selecting a station arbitrarily and analyzing the network by modifying the parameters of the selected station without repeating the analysis of the rest of the network. In section 4.3 we showed the construction of the subnetwork which is analyzed by a throughput algorithm. The subnetwork is replaced by a composite station with infinite capacity of which the service rates are set equal to the throughput values obtained by the throughput algorithm for the subnetwork. In section 4.1 we showed the phase construction of the selected station. We represented blocking delays in the selected station by phases. In order to compute the parameters of the constructed phases we introduced an iterative technique. The iteration was executed between different views of the network, i.e. the two-station network with delay phases in the selected station and each station in the originally given network as independent $M/M/1$ stations with a finite capacity. In section 4.4 we discussed the analysis of the two station network consisting of the selected station and the composite station. In section 6 we gave numerical examples and compared our results with simulation as well as with other existing techniques for blocking networks. In these examples we demonstrated the steps of our technique for networks with different topologies.
References


Parallel Computation of Performance Measures for Computer Systems

I. F. Akyildiz*, G. Bolch** and M. Paterok***

* School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
U. S. A.

** Informatik IV
*** Informatik VII
University of Erlangen-Nuernberg
8520 Erlangen
F. R. G.

Abstract
Queueing network models are used in the development and improvement of all type of hardware and software systems. In this work we extend the concept of Norton's Theorem application on queueing network models. The new concept permits a queueing network model to be partitioned into multiple disjoint subnetworks, each containing many nodes. Each subnetwork is analyzed independently, with all nodes which do not belong to the investigated subnetwork, being shortened, i.e., their mean service times are set to zero. The parallel processing of subnetworks is realized on the EGPA (Erlanger General Purpose Array) system. The succeeding convolution of subnetworks is executed in parallel by a tree convolution algorithm. Finally the exact normalization constant of the entire system is obtained. All performance measures for the originally given network or as well as any desired node can be computed in parallel from the normalization constant. This method allows to optimize the existing methods for queueing network models by computing the desired performance measures in parallel. In addition, the effect of parameter modifications on particular nodes in a network can easily be determined by this concept.

"Key Words:" Performance Evaluation, Queueing Network Models, Parallel Processing, Performance Measures, EGPA Architecture.

1. Introduction

Queueing network models are applied successfully to the performance evaluation of computer systems and communication networks. Performance measures are computed by analytical methods or simulation of queueing network models. Compared to analytical methods, simulation is a more realistic and powerful technique. However, due to the high cost of simulation and uncertain statistical accu-
racy of the results, analytical methods are the primary choice for performance analysis.

In the last two decades several analytical methods have been developed for the analysis of queueing network models. Two of the major analytical methods which have gained particular interest are the Mean Value Analysis by Reiser/Lavenberg [13] and the convolution algorithms by Busen [5] and Chandy/Hersog/Woo [7]. The Mean Value Analysis algorithm is very easy to implement and computationally fast, but it requires large memory for certain cases. The convolution algorithms require much less storage but have longer run times than those of mean value analysis.

In light of the above, the parametric analysis of Chandy, Hersog and Woo [5], where the convolution algorithm is applied, is very interesting for cases in which only one node (e.g., a CPU) in a queueing network model is to be analyzed under various system workload. It is based on an application of Norton's Theorem from electrical circuit theory to queueing networks. Chandy, Hersog and Woo [6] showed that Norton's Theorem provides an exact analysis if queueing networks have product form solution. We explain this concept by considering a closed queueing network model with K jobs, N nodes containing single, multiple and infinite servers. With this queueing network model we can construct an equivalent network in which we arbitrarily select a node N, and replace the other (N-1) nodes by a single node, called the composite node. Let \( \mu(k) \) be the composite mean service rate, where \( k \) is the number of jobs at this composite node. These composite mean service rates \( \mu(k) \), for \( k = 1,\ldots,K \), are determined by analyzing a modified version of the given network, in which the selected node has been shorted. The mean service time of that node is set equal to zero and the throughputs \( \lambda(k) \), of the shorted system for all jobs, \( k = 1,2,\ldots,K \), are calculated. These computed throughputs through the shorted network \( \lambda(k) \), are set equal to the composite mean service rates \( \mu(k) \). The solutions of the network consisting of the selected and composite nodes are identical to those of the originally given network model.

Kritsinger/van Wijk/Krzesinski [8] have extended the work of [6] to closed, open and mixed queueing networks with multiple job classes. Balsamo/Iaseolla [2], partition a network with N nodes and K jobs into two subsystems where the first subsystem contains the nodes whose behavior is to be studied and the second subsystem represents the uninteresting part, i.e., nodes whose behavior is not of interest. Their method is based on the matrix of the transition probabilities. They eliminate one uninteresting node by setting its mean service time equal to zero, and obtain a new transition proba-
bility matrix for the jobs. They repeat this elimination procedure for the next uninteresting node and construct a new transition probability matrix for the jobs. This elimination procedure must be repeated and a new transition probability matrix must be constructed for each node as they want to eliminate. This method is complex and requires a large amount of computation time for large queueing networks.

With the advances in computer and communication technology, the notions of parallel and distributed processing are gaining increased importance [9,12]. In this work we describe the architecture of the system EGPA in section 2. The concept of parallel processing of queueing network models is given in section 3. In section 4 we prove the concept. Section 5 contains the evaluation of the concept from run time and storage complexity of views. Finally section 6 contains the conclusion.

2. Erlanger General Purpose Array (EGPA) Multiprocessor System

The basic idea of EGPA is the combination of the advantages of processor arrays and usual multiprocessors working on a common memory. EGPA is a hierarchically organized multiprocessor having one controlling and administering processor for every four "working" processors. Since, in a pictorial sense, the controlling processor can be seen as being enthroned over the working processor. The EGPA system is called a pyramid as shown in Figure 1.

![Figure 1](image)

The basic elements of EGPA system are 5 processors and 5 memory blocks in which one memory block is assigned to each processor. A processor has direct access to its own block. In order to explain the communication between the processors, let us give a detailed picture in Figure 2 of the the pyramid structure given in Figure 1.
Figure 2.

Processor B has access to its own block and also to the blocks of all A processors. In other words, the B processor may access the entire memory. The A processors can only access their own blocks and blocks of two other neighbors. It is distinguished between left and right neighbors (clockwise).

The so-called diagonal accesses, e.g., from A1 to S3, are not possible.

All parts of the EGPA system are standard modules of the ATM 80-60 processors. The memory for the B-processor contains 512 Kbytes, for A-processors each memory contains 128 Kbytes.
3. Parallel Processing of Queueing Network Models on EGPA

As mentioned above, the concept of [6] involves shorting only a single node and analyzing the remaining nodes, thus reducing the originally given network to a two-node model. To short only a single node gains no real advantage in reducing the number of computations. We will extend the concept of [6], in which a queueing network model can arbitrarily be partitioned into M disjoint subnetworks, in short form SNWj (for j=1,2,...,M), each containing multiple nodes. This concept is illustrated by the example given in Figure 3.

We consider closed product form queueing networks with N nodes containing single, multiple and infinite servers, K jobs and R job classes. We partition the given network in M disjoint subnetworks. The analysis of the network is executed in three steps outlined in sections 3.1, 3.2 and 3.3, respectively.

Figure 3.
3.1. Decomposition of the Network

Figure 4 shows the given queueing network from Figure 3 decomposed into 4 subnetworks.

Each subnetwork SNW\(_j\) is analyzed by shorting all other nodes in their subnetwork, i.e., their service times are set to zero. Since the subnetworks can be analyzed independently, this analysis can be executed in parallel. As a result, the load dependent throughput values \(\lambda_{SNWj}(k)\) for each subnetwork are obtained in parallel. The computed throughput values can then be used in terms of input values for steps 2 and 3 which can be started to execute in parallel to step 1.

The independent analysis of each subnetwork is executed in parallel on the 4 A-processors. By selecting an optimal decomposition of the network the parallel processing provides an acceleration of the computations by a factor of 3 to 3.5. An optimal decomposition aggregates the entire network into multiple subnetworks such that the following relation is valid:

\[
\{\text{Number of Subnetworks}\} \mod 4 \geq 3
\]

The purpose of this relation is to prevent processors from waiting on the results of other processors.
3.2. Composing of Subnetworks

Each subnetwork $SNW_j$ containing multiple nodes can therefore be composed into a single node. The computed load dependent throughputs $\lambda_{SNW_j}(k)$ for each $SNW_j$ (for $1 \leq j \leq M$) are set equal to the load dependent service rates $\mu_{ij}(k)$ of the respective composite node, i.e.,

$$\mu_{ij}(k) = \lambda_{SNW_j}(k) \quad \text{for } j=1,\ldots,M \quad (1)$$

The composite nodes are serially connected in an arbitrary order.

![Diagram of composite nodes](image)

Figure 5.

3.3. Analysis of Serial Order

For the analysis of these arbitrarily ordered composite nodes we first compute the normalization constant. This measure is obtained by a convolution of $M$ normalization constants, each representing one subnetwork:

$$G = G_1 \ast G_2 \ast \cdots \ast G_M \quad (2)$$

where $\ast$ is the convolution operation,

$$G_j = \begin{bmatrix} G_j(0) \\ G_j(1) \\ \vdots \\ G_j(K) \end{bmatrix} \quad (3)$$

with

$$G_j(k) = \frac{G_j(k-1)}{\mu_{ij}(k)} \quad \text{for } k=1,\ldots,K \text{ and } j=1,\ldots,M \quad (4)$$

As initial value we have $G_j(0) = 1$.

The normalization constants of each subnetwork are convoluted with each other. The result of the
convolution of subnetworks $1$ to $j$ (for $j = 1, \ldots, M - 1$) is convoluted with the normalization constant of the subnetwork $(j + 1)$. Since this procedure is commutative and associative [1,11], there is no need to perform this operation in any specific order. This idea leads to a tree convolution algorithm as suggested by Lam/Lien [10,17].

A convolution of 8 subnetworks on 4 processors can be executed as follows: (with the assignment the processors to the convolution process).

![Figure 6.](image)

The ratio of the total processing time to the sequentially executed time is $3 : 8$. From the computed normalization constant we can obtain all characteristic performance measures.

As a result we obtain the normalization constant for the entire network, from which all relevant characteristic performance measures can be computed [15,18]. The measures are valid for each node of the originally given network model, Figure 3. The network given in Figure 5 can also be analyzed directly by mean value analysis for load dependent servers using the service rates $\mu_f(k)$, [14,16].
4. Proof of the Concept

Parallel analysis concept is exact if the queueing network has product form solution which will be demonstrated in the following.

A state of a queueing network with multiple job classes is given by

$$K = (K_1, \ldots, K_R)$$

is the vector of the total number of jobs in the entire network.

The state vector of the \(i\)-th node is given by:

$$\mathbf{k}_i = (k_{i1}, k_{i2}, \ldots, k_{iR})$$

A matrix \([k_n]\) represents the state \(\mathcal{S}\). For the equilibrium probability of the state \(\mathcal{S}\), the following formula is valid due to the product form solution [3]:

$$P(\mathcal{S}) = \frac{1}{G(K)} \prod_{i=1}^{N} x_i(k_i)$$  \hspace{1cm} (5)

A subnetwork, which consists only of the \(i\)-th node, has the normalization constant \(G_i\) and possesses only one state \(S\). With equation (5) it is valid that

$$x_i(k) = G_i(k)$$

for \(i = 1,2,\ldots,N\) and \(k = 0, 1, \ldots, K\)

By rewriting equation (5) we get the following product form solution:

$$P(\mathcal{S}) = \frac{1}{G(K)} \prod_{i=1}^{N} G_i(k_i)$$  \hspace{1cm} (6)

Since the sum of probabilities for all feasible states is one, it is valid that

$$G(K) = \sum_{\sum k_i - K} \prod_{i=1}^{N} G_i(k_i)$$  \hspace{1cm} (7)

The normalization constant can also be expressed analogous to equation (7) for a network consisting of composite nodes (Figure 6):

$$G(K) = \sum_{\sum k_n - K} \prod_{j=1}^{M} G_{ij}(k_{nj})$$  \hspace{1cm} (8)
or also

\[ G(K) = \sum_{k_\sigma = K} \prod_{j=1}^{M} \sum_{L_{ij} > 0} \Pi_{i \in SNW_j} G_i(L_{ij}; i) \]  \quad (9)\]

where

- \( k_{\sigma} \) is the number of jobs in the subnetwork \( SNW_j \).
- \( L_{ij} \) is the number of jobs in the \( i \)-th node of \( SNW_j \).
- \( n_{ij} \) is the total number of nodes in \( SNW_j \).

A shorted node may contain no jobs. Therefore, the only element of the normalization constant for such nodes is non-zero. \( G(0) = 1 \) and 0 is the only feasible state. Hence, we have the product over all nodes in equation \( (7) \). We obtain:

\[ G(K) = \sum_{k_\sigma = K} \prod_{j=1}^{M} \sum_{L_{ij} > 0} \Pi_{i = 1}^{N} G_{ij;i}(L_{ij}; i) \]  \quad (10)\]

By using the convolution operation we get

\[ G = G_{e_1} \times G_{e_2} \times \ldots \times G_{e_M} \]  \quad (11)\]

with

\[ G_{e_i} = G_{e_{ij}} \times \ldots \times G_{e_{ijN}} \]  \quad (12)\]

Since each node is contained in exactly one of the subnetworks, it is represented \((M-1)\) times in equation \( (11) \) by the neutral element of the convolution (due to the shorting the nodes) and exactly one time by its normalization constant.

Applying the commutative law we obtain:

\[ G = G_1 \times G_2 \times \ldots \times G_N \]  \quad (13)\]

This completes the proof.

Note that in this section we have proven mathematically that queueing networks can be decomposed into disjoint subnetworks and each subnetwork can be analyzed independently.
5. Evaluation

We do the evaluation of our concept from two different points of view: the run time and storage complexity.

5.1. Run Time

In order to measure the run times of the programs we embedded certain parameters in the program:

- CPU time measurement: This parameter measures the time of the B-processors.
- Response Time of a Job in Mean Value Analysis and in part in the Convolution
- Response Time of A Job for Computation of the Mean Number of Jobs
- Total Response Time of a Program in the System

In Table 1 we give total run time of the program on EGPA system for the analysis of a network with \( N = 6 \) nodes, \( R = 3 \) classes and \( K_r = 5 \) jobs in each class \( r = 1, 2, 3 \). We vary the number of processors (columns) and the number of subnetworks (rows). In Table 1 three different times are given where the first row shows the time for Mean Value Analysis and Convolution, the second row show the time needed for the computation of mean number of jobs and the third row shows the total response time of the program on the system.
Table 1.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.57</td>
<td>4.56</td>
<td>4.55</td>
<td>4.85</td>
</tr>
<tr>
<td>12.49</td>
<td>6.49</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>23.53</td>
<td>17.54</td>
<td>16.56</td>
<td>16.03</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14.57</td>
<td>12.57</td>
<td>12.56</td>
<td>12.56</td>
</tr>
<tr>
<td>12.49</td>
<td>6.47</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>33.61</td>
<td>25.57</td>
<td>24.08</td>
<td>23.58</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>24.54</td>
<td>22.04</td>
<td>18.23</td>
<td>17.03</td>
</tr>
<tr>
<td>12.49</td>
<td>6.50</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>43.60</td>
<td>35.08</td>
<td>29.60</td>
<td>28.22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>34.85</td>
<td>22.06</td>
<td>28.48</td>
<td>17.39</td>
</tr>
<tr>
<td>12.47</td>
<td>6.50</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>54.21</td>
<td>35.08</td>
<td>40.08</td>
<td>28.60</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>45.33</td>
<td>32.57</td>
<td>28.42</td>
<td>26.91</td>
</tr>
<tr>
<td>12.49</td>
<td>6.50</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>64.71</td>
<td>45.59</td>
<td>40.21</td>
<td>38.08</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>55.05</td>
<td>32.08</td>
<td>28.77</td>
<td>26.93</td>
</tr>
<tr>
<td>12.49</td>
<td>6.50</td>
<td>5.00</td>
<td>4.50</td>
<td></td>
</tr>
<tr>
<td>74.08</td>
<td>45.08</td>
<td>40.19</td>
<td>38.07</td>
<td></td>
</tr>
</tbody>
</table>

The interpretation of Table 1 is as follows:

i) The first column allows us to read the time for the convolution. Since only one processor computes, after mean value analysis execution of each subnetwork the obtained normalisation constant is convoluted with the previous one. For only one subnetwork there is no need for convolution. Each convolution takes 5.5 sec. on the EGPA.

ii) The time for computation of the mean number of jobs is independent of the number of subnetworks. In this case the parallel execution shows advantages.

iii) The optimal decomposition of the network as mentioned before, is dependent on the number of processors. This statement is strengthened by a comparison of the run time for MVA and Convolution for 3 and 4 processors and for 4 and 5 subnetworks. For 5 subnetworks and 4 processors one processor must execute MVA and Convolution additionally, while other processors are waiting.

iv) Mean Value Analysis (in case of one subnetwork) is faster than the convolution algorithm which is a known fact in queueing network modeling. The convolution is the most time consuming operation of the entire concept.
Several other examples have been executed and run times have been measured. The same example with $N=6$ nodes and $R=3$ job classes each with $K_r=10$ for $r=1,2,3$ jobs has been considered. Additionally an example with $N=18$ nodes, $R=3$ job classes with $K_r=10$ jobs per class has been analyzed. Each network has been decomposed into 4 subnetworks and executed on 4 A-processors of the EGPA system:

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>398.5</td>
<td>410.9</td>
</tr>
<tr>
<td>64.5</td>
<td>164.0</td>
</tr>
<tr>
<td>470.8</td>
<td>588.8</td>
</tr>
</tbody>
</table>

Table 2.

A comparison of both measurements shows clearly that the run times for Mean Value Analysis and Convolution do not differ even though the number of nodes in the second example is triple than the number of nodes in the first example. In the second example MVA has longer run time. The effort for Convolution algorithm which is independent of the number of nodes is the same for both examples. However, in computation of the mean number of jobs the large number of nodes causes longer run times.

In Table 1 there is a time difference of factor 2 between one and four computing processors. This speed-up factor will be higher if the network size gets larger, i.e., the computational effort of the parallel execution makes more sense for large queueing networks.

5.2. Storage Complexity

As generally known, the disadvantage of MVA is the large storage requirement for networks with multiple job classes. MVA needs for the analysis of a network with $N$ nodes, $K$ jobs and $R$ classes

$$O\left[N \cdot \prod_{r=1}^{R} (K_r + 1) \right]$$

storage space. Our concept reduces this amount to

$$O \left[ \prod_{j=1}^{\max \{N_j\}} (N_j + 2) \prod_{r=1}^{R} (K_r + 1) \right]$$

(15) where $N_j$ is the number of nodes in the subnetwork $j$. The constant 2 is the amount for the throughput measure and for an intermediate result of the convolution. Since we decompose the
network into \( N \) subnetworks each with single node we obtain as minimum storage space:

\[
O \left[ 3 \cdot \prod_{i=1}^{K_r} (K_r + 1) \right]
\]

(16)

This corresponds to Chandy, Herzog, Woo algorithm \([6]\) which has long run time because of several convolutions.

MVA needs large storage space for conditional marginal probabilities in case of load-dependent nodes \([14, 16]\). We need

\[
O \left[ N_m \cdot \left( \max_{i=1, \ldots, N} \{m_i\} - 1 \right) \prod_{i=1}^{K_r} (K_r + 1) \right]
\]

(17)

storage space. \( \max \{m_i\} \) is the maximum number of servers in a node. \( N_m \) is the number of multiserver nodes. The storage space of the conditional marginal probabilities takes \( (\max \{m_i\} - 1) \) times more than the storage of mean number job measures.

This amount can drastically be reduced if we put at most one multiserver node in each subnetwork:

\[
O \left[ \left( \max_{i=1, \ldots, N} \{m_i\} - 1 \right) \prod_{i=1}^{K_r} (K_r + 1) \right]
\]

(18)

6. Conclusion

The Parallel Analysis concept can be applied in order to simplify the computational requirements involved in large queueing network models. Using parallel processing, the major disadvantage of the mean value analysis algorithms (the large storage requirement) and the main disadvantage of the convolution algorithms (long run times) can be reduced radically. This is due to the fact that the large
queueing network is analyzed in parallel as multiple independent small networks.

The major advantage of this concept is that the computational expenses are reduced if some nodes from a large queueing network model are to be investigated under various system workloads. In this case we must determine the normalization constant of the subnetwork which contains the interesting nodes. These results are then convoluted with the normalization constants of the remaining subnetworks. The advantage results from the fact that the normalization constants for the remaining subnetworks are computed only once in the beginning, remaining fixed under various system workloads.

The concept of parallel processing of queueing network models is applied in order to simplify the computational requirements involved in very large queueing network models. Using parallel processing, the major disadvantage of the mean value analysis algorithms (the large storage requirement) and the main disadvantage of the convolution algorithms (long run times) can be reduced radically. queueing network is analyzed in parallel as multiple independent small networks.

The major advantage of this technique is that the computational expenses are reduced if some nodes from a large queueing network model are to be investigated under various system workloads. In this case we must determine the normalization constant of the subnetwork which contains the interesting nodes. These results are then convoluted with the normalization constants of the remaining subnetworks. The advantage results from the fact that the normalization constants for the remaining subnetworks are computed only once in the beginning, remaining fixed under various system workloads.
References


Approximate Analysis of Load Dependent General Queueing Networks

I. F. Akyildiz
Albrecht Sieber

Reprinted from
IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
Vol. 14, No. 11, November 1988
Approximate Analysis of Load Dependent General Queueing Networks

I. F. AKYILDIZ, MEMBER, IEEE, AND ALBRECHT SIEBER

Abstract—A method for approximate solutions to load dependent closed queueing networks containing general service time distributions and FCFS scheduling disciplines is presented. The technique demonstrated is an extension of the well-known method of Marle. A new formula for the conditional throughputs is derived. After each iteration a check is performed to guarantee that the results obtained are within a tolerance level ε. These iterations are repeated whenever invalid results are detected. On the average, the solutions obtained vary by less than 5 percent from their respective exact and simulation results.

Index Terms—Conditional throughputs, performance analysis, performance measures, queueing networks.

I. INTRODUCTION

Queueing networks have been used extensively in the modeling and analysis of computer systems and communication networks since the last two decades. The low computational cost and adequate accuracy of queueing network models in predicting the performance of computer systems has been generally established [12], [14], [32]. This is primarily due to their ability to model multiple independent resources and the sequential use of these resources by different jobs. The basic results of network queueing theory were presented by Jackson, Gordon and Newell, and Buzen [5], [15], [17]. They demonstrated that solutions to both open and closed queueing networks with exponentially distributed arrival and service times implement a first-come-first-served queueing discipline have a product form.

A product form implies that all stations have equilibrium state probabilities consisting of factors representing the individual stations within the network. The resulting implication is that the individual stations behave as if they were separate queueing systems. Baskett, Chandy, Muntz, and Palacios [3] extended these results to obtain product form solutions for open, closed, and mixed queueing networks with multiple job classes, nonexponential service time distributions, and different queueing disciplines.

Such queueing networks are known to have local balance [8], Markov implies Markov property [26], or the station balance property [8]. Networks containing four types of stations with queueing disciplines FCFS (first-come-first-served), PS (processor sharing), IS (infinite server), and LCFS-PR (last-come-first-served preemptive resume), have product form solution. The service times per visit in FCFS stations should have a negative exponential distribution with the same mean for all classes. The other queueing disciplines allow general service time distributions (with rational Laplace transforms).

Several extensions to the existing product form networks have been proposed in recent years. Three new station types with exponential service time distributions and the queueing disciplines: SIRO (service-in-random-order) [35], LBPS (last batch processor sharing where jobs are processed as per last-come-first-served, but the arrival time is determined for a batch of jobs, not individual jobs) [27] and WEIRDP (a portion p of the processor is allocated to the first job in the queue and the remainder (1 − p) is allocated to the remaining jobs in the queue) [9], have been shown to have product form solutions.

Product form networks can be solved efficiently using the convolution algorithms of Buzen [5] and Chandy/Herzog/Woo [6] as well as mean value analysis of Reiser and Lavenberg [29]–[31], as well as their variations [32].

Despite their popularity, several drawbacks do exist with product form networks. Probably the most significant of these is the assumptions that must be made when designing the system model. It is these assumptions that allow us to fit a given model to the format required for obtaining performance measures using product form network algorithms. Moreover, not all queueing networks will conform to one of the classes covered by product form algorithms. A queueing model containing even a single station not meeting one of the above mentioned seven basic types does not have a product form solution. This introduces problems when one considers the fact that service time distributions tend to demonstrate a high variance at CPU's (hyperexponential) and low variances at the I/O devices (Erlang). Furthermore, incorrectly assuming an exponential service time distribution can introduce significant errors into the results of performance evaluation for actual systems.

If a queueing network model is not amenable to a product form solution, it is often necessary to build and solve a Markov chain [36] which correctly accounts for the nonproduct form characteristics. However, such an approach
becomes infeasible for models requiring a Markov chain with a very large number of states. In such cases, simulation or approximate analytic techniques are required.

A large variety of classical approximation methods exist for dealing with distributions and/or scheduling disciplines not containing product form, for example, product form approximation methods such as diffusion approximation ([13], [18], [19], EPF-technique (extended product form) of Shum and Buzen [34] or aggregation/decomposition methods such as iterative approximation [7], decomposition approximation [11], method of Kuehn [21], method of Marie [22]-[24] or maximal entropy method [20], [39] or response time preservation method [1] or mean value analysis approximations [2].

It should be noted that the above list does not contain all approximation methods. It is merely provided as a framework outlining the basic methods that currently exist for approximate methods to queueing networks.

Our experience indicates that existing or suggested approximate solution methods contain inherent disadvantages or restrictions. In some cases they provide results which differ widely from the exact values. In addition, these methods are restricted to networks with load-independent stations. The service times, visit ratios and routing probabilities do not vary with job population changes. These assumptions are too rigid for many real systems. For example, if the moving-arm disk employs a scheduler that minimizes arm movement, a measurement of the mean seek time during a lightly loaded baseline period will differ significantly from the average seek time observed in a heavily loaded projection period. Similarly, the visit ratios for a swapping device will differ in baseline and projection periods having different average levels of multiprogramming. Another example is in the modeling of multiprocessors, where account must be taken of the degradation in performance due to the memory interference and software lockout (mutually exclusive access to share data structures). All of these examples illustrate the importance of considering load-dependent behavior.

Analytical methods can deal with only two kinds of load-dependent behavior:

1) A station’s service function may depend on the length of that station’s queue [33].
2) The visit ratios and transition probabilities may depend on the total number of jobs in the system [38].

We will consider load-dependency in which the service time of each station is distributed with load-dependent mean value 1/ \( \mu_i(k) \) (for \( k = 1, \ldots, K \)) and general distribution function \( F_i(t) \) having a rational Laplace transform.

3) Each station has FCFS scheduling discipline and infinite capacity.
4) A job serviced by station \( i \) proceeds to station \( j \) with probability \( p_{ij} \) for \( i, j = 1, 2, \ldots, N \).

II. APPROXIMATE ANALYSIS OF LOAD DEPENDENT NETWORKS

We consider closed queueing networks with the following characteristics:

1) There are \( N \) stations and \( K \) single class jobs.
2) The service time of each station is distributed with load-dependent mean value 1/ \( \mu_i(k) \) (for \( k = 1, \ldots, K \)) and general distribution function \( F_i(t) \) having a rational Laplace transform.
3) Each station has FCFS scheduling discipline and infinite capacity.

A. Load Dependent Arrival Rates

To determine the load-dependent arrival rates \( \lambda_i(k) \) (for \( k = 0, 1, 2, \ldots, K \)) of a station \( i \) (for \( i = 1, \ldots, N \)), the \( i \)th station is shorted, i.e., its service time is set to zero as shown in Fig. 1.

It is assumed that the subnetwork satisfies local balance and has product form solution. The throughput values \( \lambda'_i(k) \) of the subnetwork can be obtained by any product form network algorithm such as mean value analysis [31] or convolution algorithms [5], [6], [33] for load-dependent networks.

The load-dependent arrival rates \( \lambda_i(k) \) to the \( i \)th station are then:

\[
\lambda_i(k) = \lambda'_i(K - k) \quad \text{for} \quad k = 0, 1, \ldots, K - 1. \tag{1}
\]

It is clear that if \( k \) jobs are present at the \( i \)th station, then \((K - k)\) jobs remain in the subnetwork. Thus the throughput of the subnetwork with \((K - k)\) jobs is equal to the arrival rate of the \( i \)th station with \( k \) jobs. Note that

\[
\lambda_i(K) = 0 \tag{2}
\]

since no job is in the subnetwork and consequently the throughput will be zero, \( \lambda'_i(0) = 0 \).

In this way each station is shorted and the throughput \( \lambda'_i \) of the corresponding subnetwork is computed and assigned to the according arrival rates \( \lambda_i \).
Note that in the first iteration the service rates of each station are the originally given values \( \mu_i \). In future iterations the conditional throughputs \( v_i(k) \), computed in Section II-B, are used as the adjusted service rates for the stations.

### B. Load Dependent Service Rates

We have assumed that the service times follow an arbitrary distribution with a rational Laplace transform. Cox [10] demonstrated that any distribution with rational Laplace transform can be represented by a sequence of fictitious phases as shown in Fig. 2.

Each time a job completes a phase it may either depart from the station or it may proceed to the next phase. The total time a job spends in a phase is exponentially distributed. In general, each phase has a different mean service rate. Let \( \mu_{ij}(k) \) denote the load-dependent service rate at the \( j \)th phase (\( j = 1, 2, \cdots, n \)) of the \( i \)th station (\( i = 1, \cdots, N \)). Also, let \( a_i \) be the probability that a job upon completion of its service at the \( j \)th phase will proceed to the \( (j+1) \)th phase. \( b_j \) denotes the probability that a job upon completion of its service at phase \( j \) will depart from the station. This type of service distribution is known as the Coxian distribution and it is denoted by \( C_n \) where \( n \) is the number of phases. Jobs are assumed to arrive at the station in a Poisson fashion at the load-dependent rate \( \lambda(k) \) (computed in Section II-A). This type of station has the shorthand notation \( \lambda(k)/C_n/1 \) - FCFS. Cox's representation of arbitrary distributions with rational Laplace transforms is useful when dealing with nonexponential distributions. In the modeling of nonexponential service time distributions we have an estimation of the first and second moments of the service law, i.e., the expected time distributions we have an estimation of the first and second moments of the service law. Marie [22] has shown that for any mean \( \mu \) (first moment) and any squared coefficient of variation \( (c^2 = \text{variance}/\mu^2) \) (second moment) such that \( 0.5 \leq c^2 \leq \alpha \), it is possible to represent a station's server by a Cox-model with two phases.

The parameters \( \mu_{11}(k), \mu_{12}(k), a_i \) and \( b_i \) are determined as follows [24]:

\[
\mu_{11}(k) = 2\mu_i(k) \quad (3a)
\]

\[
\mu_{12}(k) = \frac{\mu_i(k)}{c_i^2} \quad (3b)
\]

\[
a_i = \frac{1}{2c_i^2} \quad (3c)
\]

\[
b_i = 1 - a_i. \quad (3d)
\]

Note that for values of \( c_i \) less or equal to 0.5, Marie [24] suggested an Erlang type of distribution.

### Various Studies

Various studies of single server and multiple server stations with Coxian distributions have been reported in the literature. These studies concentrated on the derivation of the probability distribution of the number of jobs in the system using various recursive procedures. In particular, Herzog, Woo, and Chandy [16] and Bux and Herzog [3] obtained numerical results for a single server station with state dependent arrival and service rates, and assuming that the interarrival times as well as the service times follow a Coxian distribution.

Marie [24] studied the queue length probability distribution of a single server station with a Coxian service time distribution and exponentially distributed load-dependent interarrival times. His approach is based on the notion of the conditional throughput \( v_i(k) \) (adjusted service rate) which is obtained using a recursive formula. Marie's model is extended to multiple servers by Stewart and Marie [37] and Marie, Snyder, and Stewart [25] using numerical techniques. Perros [28] gives exact closed-form expression of the probability distribution of the number of jobs in an \( M/C_n/1 \) station.

In this section we derive the formulas for conditional throughputs \( v_i(k) \) (for \( k = 1, \cdots, K \)) (adjusted service rates) for load-dependent networks in detail, since the classical method of Marie [22]-[24] differs from the technique presented only in the computation of \( v_i(k) \).

A state of a station is denoted by a pair:

\[
k = (k, j)
\]

where

- \( k \) is the number of jobs, \( k = 0, 1, 2, \cdots, K \)
- \( j \) is the number of phases, \( j = 1, 2, \cdots \).

A transition from one state to another takes place either when a new job arrives or when a job leaves the station through either phase. We will represent arrival rates by \( \lambda_i(k) \), where the arrivals rates are dependent on the number of jobs in the station as computed in Section II-A. A job leaving the station after phase one is denoted by \( b_i \mu_i(k) \). \( \mu_{12}(k) \) is the departure rate of a job leaving the station after phase two. A job enters into phase two at a rate \( a_i \mu_{11}(k) \) and may do so only after receiving service in phase one. The state \( (0) \) denotes that no job is in the \( i \)th station.

To compute the probability of being in a state as shown in the state transition diagram, Fig. 3, we use the Chapman–Kolmogorov equations. Consider the probability \( p_i(k, j) \) as the probability of being in the \( i \)th station of the network given \( k \) jobs and the current job in phase \( j \). There are six cases to consider.
Let the probability of $k$ jobs in a station, independent of which phase a job is executing in, be denoted as

$$p_i(k) = p_i(1, 1) + p_i(1, 2)$$ for $k = 1, \ldots, K$. 

(10)

We can express the conditional throughputs $v_i(k)$ (adjusted service rates) in terms of the equilibrium state probabilities and the service rates of the Cox phases. Note that the service rates of the Cox phases (3) do not vary in the entire execution of the algorithm:

$$v_i(k) = \frac{1}{p_i(k)} \left[ p_i(k, 1) b_i \mu_1(k) + p_i(k, 2) \mu_2(k) \right].$$

(11)

The derivation of (11) is obvious in Fig. 4.

The following theorem has been proven by Marie [22]:

$$p_i(k - 1) \lambda_i(k - 1) = p_i(k) v_i(k).$$

(12)

Simply stated, the probability that a job leaves a station in which there are $k$ jobs, is equal to the probability that a job arrives at the same station when there are $(k - 1)$ jobs.

To determine $v_i(k)$ we modify (7) and obtain:

$$p_i(k, 1)\left[ \mu_1(k) + \lambda_i(k) \right] = p_i(k - 1, 1) \lambda_i(k - 1) + p_i(k + 1, 1) \lambda_i(k + 1) + p_i(k, 1) \mu_2(k + 1) \lambda_i(k + 1) \mu_2(k + 1) \lambda_i(k + 1)$$

(5)

From (10) and (11) we obtain

$$v_i(k) = \frac{p_i(k, 1) b_i \mu_1(k) + p_i(k, 2) \mu_2(k)}{p_i(k, 1) + p_i(k, 2)}$$

(15)

which can be rewritten as

$$\frac{p_i(k, 1)}{p_i(k, 2)} = \frac{\mu_2(k) - v_i(k)}{v_i(k) - b_i \mu_1(k)}.$$
\[
\frac{\mu_2(k) - v_i(k)}{v_i(k) - b_i\mu_1(k)} = \frac{\lambda_i(k) + \mu_2(k)}{a_i\mu_1(k)} - \frac{\lambda_i(k - 1)}{a_i\mu_1(k)} \cdot \frac{p_i(k - 1, 2)}{p_i(k, 2)}. \tag{17}
\]

\[
v_i(k) = \frac{[\mu_2(k - 1) - b_i\mu_1(k - 1)][\mu_1(k)\mu_2(k + b_i\mu_1(k)\lambda_i(k)]]}{[[\lambda_i(k) + \mu_2(k) + a_i\mu_1(k)[\mu_2(k - 1) - b_i\mu_1(k - 1)]] - \{[v_i(k - 1) - b_i\mu_1(k - 1)][\mu_2(k) - b_i\mu_1(k)]]}. \tag{23}
\]

To solve the (17) in terms of \(v_i(k)\), without determining the state probabilities \(p_i(k, j)\) the term

\[
\left[ \frac{p_i(k - 1, 2)}{p_i(k, 2)} \right]
\]

must be replaced in (17).

Observe that

\[
\frac{p_i(k - 1, 2)}{p_i(k, 2)} = \frac{p_i(k - 1, 2)}{p_i(k - 1)} \cdot \frac{p_i(k - 1)}{p_i(k)} \cdot \left[ 1 + \frac{p_i(k, 1)}{p_i(k, 2)} \right]. \tag{18}
\]

From (11) we obtain

\[
p_i(k - 1) v_i(k - 1) = p_i(k - 1, 1) b_i\mu_1(k - 1) + p_i(k - 1, 2) \mu_2(k - 1). \tag{19}
\]

Rewriting

\[
p_i(k - 1) v_i(k - 1) + p_i(k - 1, 2) b_i\mu_1(k - 1)
= p_i(k - 1, 1) b_i\mu_1(k - 1)
+ p_i(k - 1, 2) \mu_2(k - 1). \tag{20}
\]

From (20) the following equation can be derived:

\[
\frac{p_i(k - 1, 2)}{p_i(k - 1)} = \frac{v_i(k - 1) - b_i\mu_1(k - 1)}{\mu_2(k - 1) - b_i\mu_1(k - 1)}. \tag{21}
\]

Substituting (12), (16), and (21) in (18) the term for equilibrium state probabilities in (17) can be expressed as follows:

\[
\frac{\mu_2(k) - v_i(k)}{v_i(k) - b_i\mu_1(k)} = \frac{\lambda_i(k) + \mu_2(k)}{a_i\mu_1(k)} - \frac{\lambda_i(k - 1)}{a_i\mu_1(k)} \cdot \frac{v_i(k - 1) - b_i\mu_1(k - 1)}{\mu_2(k - 1) - b_i\mu_1(k - 1)} \cdot \frac{v_i(k)}{\lambda_i(k - 1)} \cdot \left[ 1 + \frac{\mu_2(k) - v_i(k)}{v_i(k) - b_i\mu_1(k)} \right]. \tag{22}
\]

The solution of (22) after \(v_i(k)\) provides the desired formula for the recursive computation of the conditional throughputs \(v_i(k)\) (adjusted service rates) from the parameters of the Cox-2-distribution and the arrival rates \(\lambda_i(k)\) for \(k > 1\).

The computation for \(v_i(k)\) is an iteration process based on previous \(v_i(k)\) values. Analogous to (6) we obtain:

\[
p_i(1, 1) = \frac{\lambda_i(1) + \mu_2(1)}{a_i\mu_1(1)}. \tag{24}
\]

Equation (16) is also valid for \(k = 1\). Substituting (16) in (24) we obtain:

\[
\frac{\lambda_i(1) + \mu_2(1)}{a_i\mu_1(1)} = \frac{\mu_2(1) - v_i(1)}{v_i(1) - b_i\mu_1(1)}. \tag{25}
\]

Solving (25) yields:

\[
v_i(1) = \frac{b_i\lambda_i(1)\mu_i(1) + \mu_1(1)\mu_2(1)}{\lambda_i(1) + \mu_2(1) + a_i\mu_1(1)}. \tag{26}
\]

Note that in case of load-independent stations, it is valid that

\[
\mu_i(k) = \mu_i \quad \text{for} \quad i = 1, \ldots, N; \quad k = 1, \ldots, K;
\]

\[
j = 1, 2. \tag{27}
\]

Substituting (27) in (23) the following formula for conditional throughputs \(v_i(k)\) for load-independent networks is obtained, which is also given by Marie [24].

\[
v_i(k) = \frac{b_i\lambda_i(k)\mu_i(1) + \mu_1(1)\mu_2(1)}{\lambda_i(k) + \mu_1(1) + \mu_2(1) - v_i(k - 1)}. \tag{28}
\]

### C. Equilibrium State Probabilities

From (12) and considering that

\[
\sum_{k=0}^{K} p_i(k) = 1
\]

the equilibrium state probabilities \(p_i(k)\) for \(k = 0, 1, \ldots, K\) are obtained as follows [23]:

\[
p_i(k) = p_i(0) \prod_{n=0}^{k-1} \frac{\lambda_i(n)}{v_i(n + 1)} \quad \text{for} \quad k = 1, \ldots, K \tag{29}
\]

where

\[
p_i(0) = \left[ 1 + \sum_{k=1}^{K} \prod_{n=0}^{k-1} \frac{\lambda_i(n)}{v_i(n + 1)} \right]^{-1}. \tag{30}
\]
D. Termination Test

After each iteration we check to see if the sum of the mean number of jobs is equal to the total number of jobs in the given network within a tolerance level \( \varepsilon \):

\[
\left| K - \sum_{i=1}^{N} p_i(k) \right| < \varepsilon. \tag{31}
\]

Additional check is made to see if the throughput rates of each station are consistent with the topology of the network:

\[
\left| \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_i(k) v_i(k) - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_i(k) v_i(k) \right| < \varepsilon
\]

\[
\text{for } j = 1, \cdots, N \tag{32}
\]

where \( e_i \) is the mean number of visits that a job makes to station \( i \) and is compared by:

\[
e_i = \sum_{j=1}^{N} e_j p_{ji}
\]

and \( \varepsilon \) is a tolerance level. Usual value is \( \varepsilon = 10^{-4} \).

If one or both of these conditions are violated, the service rates are adjusted

\[
\mu_i(k) := v_i(k) \quad \text{for } k = 1, \cdots, K, \tag{33}
\]

i.e., the conditional throughputs [(25) and (28)] are assumed to be the new service rates, and the next iteration is carried out. Iterations continue until acceptable tolerances are obtained. The performance measures can then be computed by the following formulas.

- Throughput of each station:

\[
\lambda_i(K) = \sum_{k=1}^{K} p_i(k) \mu_i(k). \tag{34}
\]

- Mean number of jobs in each station:

\[
\bar{K}_i(K) = \sum_{k=1}^{K} k \cdot p_i(k). \tag{35}
\]

- Mean residence time:

\[
\bar{t}_i(K) = \frac{\bar{K}_i(K)}{\lambda_i(K)}. \tag{36}
\]

III. Algorithm Summary

The following is the complete algorithm for calculating performance measures for the load-dependent general networks:

1) Determine the parameters for the Cox distribution using (3).

2) Iterative Part:

a) Compute the throughput values \( \lambda_i'(k) \) for each subnetwork without the \( i \)th station (for \( i = 1, \cdots, N \)) using the convolution algorithm [5], [6], [33] or mean value analysis for load dependent stations [31]. Assign the throughput values \( \lambda_i'(k) \) with \( k \) jobs to the arrival rates of the \( i \)th station with \( (K-k) \) jobs (1). 

b) Compute the conditional throughputs \( v_i(k) \) using (23) and (26) for \( i = 1, \cdots, N \) and \( k = 1, \cdots, K \).

c) Compute the equilibrium state probabilities for stations \( i = 1, \cdots, N \) using (29) and (30).

d) Check the termination conditions (31) and (32). If the test is not successful, then apply (33) and goto a). Otherwise compute the performance measures from (34)–(36).

IV. Evaluation

The algorithm has been implemented on a VAX 11/780 system. The computation of load-dependent arrival rates, in Section II-A has been realized by both the convolution algorithm [5], [6], [33] and also mean value analysis for load dependent networks [31]. Our tests have shown that the mean value analysis for load-dependent networks has the advantage of handling a greater number of jobs. Several different networks containing two to ten stations were analyzed, with the number of jobs ranging from ten to eighty in each network. The termination value \( \varepsilon \) ranged from \( 10^{-4} \) to \( 10^{-6} \). The vast majority of the variations beneath 5 percent. It is clearly evident that this approximation method is capable of accurate modeling of load dependency. It should be noted that in all instances where the algorithm showed a relatively high deviation from the actual results (over 8 percent), the numbers involved were quite small. In such cases the relative error might appear large even though the difference in the two numbers is insignificant. Under these circumstances, the relative error cannot be considered a reliable indicator of the accuracy of the method.

The number of iterations in the method is not predictable. It depends on the number of stations, the number of jobs, the complementary subnetworks and the epsilon value used. In most of the cases the method converges in 6–10 iterations. Although no mathematical proof for convergence is given here or also in [22]–[24] we were unable to find any model with load dependent stations in which the method did not converge.

When analyzing the complexity of the algorithm we find that the space complexity of the algorithm is \( O(3NK) \). The majority of this space was required for the computation of the equilibrium state probabilities (29), (30). Analyzing the time complexity of the method shows that it increases drastically as the number of jobs \( K \) in the network increases.

In the following we give nine examples with different input parameters. In Examples 1–5 networks with single, multiple, and infinite servers are analyzed. In Examples 6 and 7 we treat networks with load-dependent service rates. Approximate, exact, or simulation results for performance measures such as throughput and the mean number of jobs are given in tables. The approximate results are compared with exact values obtained by numerical analysis [36] in case of small number of jobs. How-
however, numerical analysis cannot be applied for large number of jobs such as in Examples 8 and 9. The approximation is validated by simulation in those cases. The simulation results are obtained within 95 percent confidence interval. The tables also contain relative deviations which are computed by:

\[ \delta = \left( \frac{\text{Exact (or Simulation)} - \text{Approximation}}{\text{Exact (or Simulation)}} \right) \times 100. \]

In Example 3 we plot the marginal probabilities as a function of the number of jobs in each station. In the graphs, the solid lines show approximate results, the dashed lines show exact results and the dotted lines show the case where the service time distributions are assumed to be exponential (i.e., \( c_i = 1 \) for all \( i \)).

In Example 4 we calculate the performance measures for different \( c_i^2 \) parameters. We plot total throughput results for different squared coefficient of variation values in Example 5.

**Example 1**: (Tandem Network); \( K = 5 \) jobs.

<table>
<thead>
<tr>
<th>Station</th>
<th>( \mu_i )</th>
<th>( m_i )</th>
<th>( c_i^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>9</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Example 2**: \( K = 9 \) jobs.

<table>
<thead>
<tr>
<th>Station</th>
<th>( \mu_i )</th>
<th>( m_i )</th>
<th>( c_i^2 )</th>
<th>( \lambda_1 )</th>
<th>( \lambda_2 )</th>
<th>( \delta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>9</td>
<td>0.0</td>
<td>2.1084</td>
<td>2.1378</td>
<td>0.54</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
<td>10</td>
<td>0.5</td>
<td>1.7023</td>
<td>1.7288</td>
<td>0.62</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>23</td>
<td>0.5</td>
<td>1.8008</td>
<td>1.7803</td>
<td>0.71</td>
</tr>
</tbody>
</table>

**Example 3**:

<table>
<thead>
<tr>
<th>Station</th>
<th>( \mu_i )</th>
<th>( m_i )</th>
<th>( c_i^2 )</th>
<th>( \lambda_1 )</th>
<th>( \lambda_2 )</th>
<th>( \lambda_3 )</th>
<th>( \delta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>3</td>
<td>0.8</td>
<td>0.0</td>
<td>0.7</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>2</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

**Example 4**: (Tandem Network); \( K = 10 \) jobs; \( c_i^2 \) are variable.

<table>
<thead>
<tr>
<th>Station</th>
<th>( \mu_i )</th>
<th>( m_i )</th>
<th>( c_i^2 )</th>
<th>( \lambda_1^2 )</th>
<th>( \lambda_1^2 )</th>
<th>( \delta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>0.2</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0.3</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0.5</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( c_1^2 )</th>
<th>( c_2^2 )</th>
<th>( c_3^2 )</th>
<th>( \lambda_1^2 )</th>
<th>( \lambda_2^2 )</th>
<th>( \delta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.830</td>
<td>1.825</td>
<td>0.07</td>
</tr>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
<td>1.964</td>
<td>1.970</td>
<td>0.01</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>2.108</td>
<td>2.110</td>
<td>0.00</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>2.061</td>
<td>2.069</td>
<td>0.09</td>
</tr>
</tbody>
</table>

**Example 5**:

<table>
<thead>
<tr>
<th>Station</th>
<th>( \mu_i )</th>
<th>( m_i )</th>
<th>( c_i^2 )</th>
<th>( \lambda_1^2 )</th>
<th>( \lambda_2^2 )</th>
<th>( \delta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>0.2</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0.3</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0.5</td>
<td>3.532</td>
<td>3.753</td>
<td>0.08</td>
</tr>
</tbody>
</table>
Example 5: (Tandem Network); $K = 6$ jobs; $c_1^2 = c_2^2 = c_3^2$.  

Example 6: (Tandem Network); $K = 7$ jobs; $c_1^2 = 0.5$; $c_2^2 = 16$; $c_3^2 = 4$.  

Example 7: (Central Server Model); $K = 8$ jobs.  

Example 8: $K = 50$ jobs.  

Example 9: (Tandem Network); $K = 50$ jobs.  

REFERENCES  


I. F. Akyildiz (M'86) was born in Istanbul, Turkey, in 1954. He received the Vordiplom, Diplom Informatiker, and Doctor of Engineering degrees in computer science from the University of Erlangen—Nuremberg, West Germany, in 1978, 1981, and 1984, respectively.

From 1981 through 1985 he served as a Scientific Employee with the Department of Infor- matik IV (Operating Systems) at the University of Erlangen—Nuremberg. During that time he coauthored a textbook entitled Analysis of Computer Systems (in German) published by Teubner-Verlag in the Fall of 1982. From 1985 through 1987 he was an Assistant Professor with the Depart- ment of Computer Science at Louisiana State University, Baton Rouge. In the Fall of 1987 he joined the faculty in the School of Information and Computer Science at Georgia Institute of Technology, Atlanta, as an As- istant Professor. His research interests are performance evaluation of computer networks and distributed systems, in particular, analytical modeling and simulation, and operating systems, in particular, protection and security.

Dr. Akyildiz is a member of the Association for Computing Machinery (Sigops and Sigmetrics), GI (Gesellschaft fuer Informatik), and MMB (German Interest Group in Measurement, Modeling, and Evaluation of Computer Systems).

Albrecht Sieber was born in Erlangen, West Ger- many, in 1961. He received the Diplom Infor- matiker degree in computer science from the Uni- versity of Erlangen—Nuremberg in 1987.

From 1986 through 1987 he spent a year at Louisiana State University, Baton Rouge, as a Re- search Assistant. Currently he is a Research As- sistant with the Department of Informatik VII (Computer Architecture and Traffic Theory) at the University of Erlangen—Nuremberg. His doctoral research is focused on performance evaluation of multiprocessors and communication networks.
Analysis of Queueing Networks with Rejection Blocking

I. F. Akyildiz

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

Abstract

In this work closed queueing networks with rejection blocking are analyzed. In this type of networks blocking event occurs when upon completion of its service of a particular station's server, a job attempts to proceed to its next station. If, at that moment, its destination station is full, the job is rejected. The job goes back to the server of the source station and immediately receives a new service. This is repeated until the next station releases a job and a place becomes available. The known exact product form solution for equilibrium state probabilities is presented for closed rejection blocking networks which have reversible routing. Algorithms are given for computation of performance measures in reversible networks with rejection blocking. Nonreversible queueing networks with rejection blocking are analyzed in the second part of this work. The analysis is based on the duality concept. The number available spaces are assumed to be the holes which move in the opposite direction in the dual network. This concept provides exact product form solution for nonreversible networks with rejection blocking. Based on this solution performance measures can easily be computed.

"Key Words:" Performance Evaluation, Queueing Networks, Blocking, Equilibrium State Probabilities

This work was supported in part by the Airforce Office of the Scientific Research under Grant No. AFOSR-87-0180.

1. Introduction

Queueing networks have a great popularity as models of computer systems since 1971, because they allow the modeling of multiple independent resources such as CPU's and I/O devices and the sequential use of these resources by different jobs. The basic results of queueing network theory were given by Jackson, Gordon/Newell and Buzen [5,9,11] where they showed that the solution of open and closed queueing networks with single job class, exponentially distributed arrival and service times, FCFS queueing disciplines at all stations have product form which means that the equilibrium state probabilities consist of factors representing the states of the individual stations. This result implies that the individual stations behave as if they were
separate queueing systems. Baskett, Chandy, Muntz and Palacios [4] extended the results of
[5,7,11] and also obtained product form solution for open, closed and mixed queueing networks
with different job classes, nonexponential service time distributions and different queueing dis-
ciplines such as FCFS, RR-PS, LCFS-PR.

Product form queueing networks (also known as BCMP or separable networks) have
proved valuable in modeling a variety of computer and communication systems, and have been
flexible enough to represent adequately many of the features arising in such applications. They
have not, however, been able to provide much insight into the phenomenon of blocking,
because all algorithms for product form networks are based on the assumption that the stations
have infinite capacities.

In recent years there has been a growing interest in the development of computational
methods to analyze the blocking networks. In this work we analyze rejection blocking queueing
networks in which blocking occurs when a job completing service at station i's server and want-
ing to join station j, whose capacity is full. The job is rejected by station j. That job goes back
with a certain probability (rejection probability) to station i's server and receives a new service
with the same mean service time. This activity is repeated until station j releases a job, and a
place becomes available.

The "rejection blocking" type has been used to model systems such as communication net-
works, computer systems with limited multiprogramming, production lines and flexible
manufacturing systems. Most of the previous work was done on the "rejection" blocking in
both open and closed queueing networks. Within this category, the studies fall into two groups.
The first group provides exact results for closed queueing networks [2,3,9,10,16,19,20].
Balsamo/Iaseoalla [3], Hordijk/VanDijk [9,10], VanDijk/Tijms [19], Pittel [16] and
Yao/Buzacott [20] have shown the existence of product form solutions for closed queueing net-
works which satisfy one of the three conditions: either the network routing matrix is reversible
or the probability of blocking is constant (i.e., independent of the number of jobs in the station
causing the blocking event) or the service rate of each station is constant, provided that it is
impossible to have empty station at any time.

The second group of studies is characterized by the specific solution method \[1,6,13\]. Caseau and Pujolle [6] studied a queueing network consisting of two or more stations in tandem with blocking with a view to obtaining an approximate expression of the maximum throughout. A job upon completion of its service at the \(i\)-th server proceeds to join the \((i+1)\)-st station. If this station is full, the job is not accepted and it simply returns to the previous server where it receives another service. This process is repeated several times until the job is eventually admitted to the \((i+1)\)-st station. This blocking process arises in packet switching systems.

Several other investigators in recent years have published results on queueing networks with rejection blocking. A bibliography of studies about queueing network models with all types of blocking is given by Perros [15].

2. Model Description

We analyze closed queueing networks with \(N\) stations and \(K\) jobs which constitute a single job class. Each station has single server and each server has an exponentially distributed service time with mean value \(1/\mu_i\) (for \(i = 1,2,...,N\)). Each station has a fixed finite capacity, \(M_i\), where \(M_i = \text{queue capacity} + 1\). A job which is serviced by station \(i\) proceeds to station \(j\) with the transition probability \(p_{ij}\) (for \(i,j = 1,2,...,N\)), if station \(j\) is not full. In other words, the number of jobs in station \(j\), \(k_j\), is less than \(M_j\). Otherwise, the job will be rejected from station \(j\), and it will return to the server of station \(i\) and will receive another round of service. This is repeated until a place is available in station \(j\). Furthermore, we assume that

\[
K < M
\]

which means that the total number of jobs, \(K\), in the network may not exceed the total capacity

\[
M = \sum_{i=1}^{N} M_i
\]

of the entire network. The service discipline of each station is First-Come-First-Served.
Section 3 explores the product form solution for reversible queueing networks given by Hordijk and van Dijk [9]. In section 4 we introduce two algorithms for computation of \( G(K) \). In section 5 we derive new formulas for performance measures for reversible queueing networks. Section 6 and 7 contain the analysis of nonreversible queueing networks.

3. Product Form Solution for Reversible Queueing Networks

A queueing network is "reversible" [12,14] if the following condition is satisfied:

\[
e_i \ p_{ij} = e_j \ p_{ji} \quad \text{for all } i,j = 1,2,...,N.
\]

(1)

This equation states that the rate at which jobs arrive at station \( j \) from station \( i \) equals the rate at which jobs leave station \( j \) to return to station \( i \). Simple examples for reversible networks are two-station networks and central server models.

It is well known that there exists a positive solution for \( e_i \):

\[
e_i = \sum_{j=1}^{N} e_j \ p_{ji} \quad \text{for } i = 1,2,...,N
\]

(2)

The relative utilization (also called loadings) of station \( i \) is denoted by \( z_i \) and is computed by

\[
z_i = e_i / \mu_i \quad \text{for } i = 1,2,\ldots,N
\]

(3)

Let \( k_i \) denote the number of jobs at station \( i \) (in waiting and in service) and the state of the network is defined as the vector

\[k = (k_1, k_2, \ldots, k_N)\]

A state \( k \) is feasible, if \( k_i \leq M_i \) for \( i = 1,\ldots,N \), where \( k_i \) denotes that there \( k \) jobs in station \( i \).

Hordijk and VanDijk [9] have shown that in a closed queueing network with rejection blocking satisfying the reversibility condition (1), the equilibrium probability distribution for a feasible state \( (k) \) is given by the following product form of marginal probabilities:

\[
p(k_1, k_2, \ldots, k_N) = \frac{1}{G(K)} \prod_{i=1}^{N} z_i^{k_i} \delta_i(k_i)
\]

(4)
where \( G(K) \) is the normalization constant with the property to make all the probabilities sum to one and the binary function \( \delta_i(k_i) \) is given by

\[
\delta_i(k_i) = \begin{cases} 
0 & \text{if } k_i > M_i \\
1 & \text{if } k_i \leq M_i 
\end{cases}
\]

which eliminates nonfeasible states that exceed the station capacities.

In order to compute the equilibrium state probabilities, the normalization constant \( G(K) \) must be determined. A naive technique to calculate \( G(K) \) is to enumerate all states and compute their relative probabilities. Absolute probabilities can then be determined from the relative probabilities by normalizing their sum to one. This is feasible only for small networks, because the number of states grows rapidly in larger queueing networks. An efficient algorithm to determine \( G(K) \) without enumerating all states of the network was first introduced by Buzen [5]. In the mean time several algorithms were developed for computation of the normalization constant and performance measures. All methods are based on the assumption that the stations have infinite capacities. None of the algorithms mentioned above can be applied on blocking queueing networks.

In the following section we introduce two algorithms for computation of the normalization constant in queueing networks with rejection and reversible routing.

4. Algorithms for Computation of the Normalisation Constant

a) Convolution Algorithm for Blocking Queueing Networks

We define an auxiliary function \( g(k,n) \), where the first parameter \( k \) denotes the number of jobs and the second parameter \( n \) the number of stations in the network. The algorithm is executed as follows:

The initial values are:

\[
g(0,n) = 1 \quad \text{for } n = 1, 2, ..., N
\]

For \( k = 1, ..., K \) compute

\[
g(k,1) = \begin{cases} 
s^k & \text{if } k \leq M_1 \\
0 & \text{if } k > M_1 
\end{cases}
\]
The auxiliary function \( g(k, n) \) is given by:

\[
g(k, n) = g(k, n-1) + x_n g(k-1, n)
\]

for \( n = 1, 2, ..., N \) and \( k = 1, 2, ..., M_n \).

If the capacity of the \( n \)-th station \( M_n \) is exceeded, i.e. for \( k > M_n \), \( g(k, n) \) is determined as follows:

\[
g(k, n) = g(k, n-1) + x_n g(k-1, n) - z_n^k h(l, n)
\]

The function \( h(l, n) \) eliminates the nonfeasible states. It is given by:

\[
h(l, n) = \frac{1}{l} \sum_{i=1}^{n-1} \sum_{j=1}^{i} x_i^j h(l-j, n) \quad \text{for} \quad l = 0, 1, ..., K
\]

with initial value \( h(0, n) = 1 \) for \( n = 1, 2, ..., N \).

The objective of the algorithm is the computation of the value of the function \( g(K, N) \) which corresponds to the normalization constant \( G(K) \).

\( b) \) **LBANCBLO (Local Balance Algorithm for Normalising Constant in Blocking Queueing Networks)**

Based on Mean Value Analysis (MVA) of Reiser and Lavenberg [17], Sauer and Chandy [18] developed an algorithm LBAN which is valid only for closed product form networks. For closed queueing networks with finite station capacities the LBAN algorithm cannot be applied in its suggested form.

LBANCBLO uses the following recursion formula, in order to determine the unnormalized mean queue lengths of the stations:

\[
q_i(k) = x_i \left[ G(k-1) + q_i(k-1) - z_i^{M_i} (M_i+1) y_i(l) \right]
\]

with initial values:

\[
G(0) = 1 \quad \text{and} \quad q_i(0) = 0 \quad \text{for} \quad i = 1, 2, ..., N
\]

The function \( y_i(l) \) eliminates the non-feasible states:

\[
y_i(l) = \begin{cases} 
\frac{1}{l} \sum_{j=1}^{N} x_j \left[ G(j-1) + q_j(j-1) - z_j^{M_j} (M_j+1) y_j(l) \right] \\
0 & \text{if} \quad l > M_i \\
\text{if} \quad l \leq M_i
\end{cases}
\]

where
\[ s_j(l) = x_j \cdot y_j(l-1) + s_j(l-1); \quad \text{for} \quad j = 1, 2, \ldots, N \]

for \( l = 0, 1, 2, \ldots, K \) and with the initial values \( s_2(0) = 0 \) and \( y_1(0) = 1 \).

The normalization constant is then obtained by:

\[ G(K) = \frac{1}{K} \sum_{i=1}^{N} q_i(K) \quad (10) \]

5. Performance Measures

As mentioned before the classical formulae [18] cannot be utilized for computation of performance measures in queueing networks with blocking.

The marginal state probabilities, i.e., the probabilities that there are \( l \) jobs in the \( i \)-th station (for \( i = 1, \ldots, N \)) is given by:

\[ p_i(l) = \begin{cases} \frac{z_i}{G(K)} \cdot \frac{G_i-(K-l)}{G_{i-1}} \quad & \text{for} \quad l = 1, \ldots, M_i \\ 0 & \text{for} \quad l = M_i+1, \ldots, K \end{cases} \quad (11) \]

where \( G_i \) is the normalization constant computed without considering the \( i \)-th station.

The mean number of jobs in the \( i \)-th station is given by:

\[ E_i = \sum_{l=1}^{M_i} l \cdot p_i(l) \quad \text{for} \quad i = 1, \ldots, N \quad (12) \]

By substituting equation (11) in equation (12) we obtain:

\[ E_i = \frac{1}{G(K)} \sum_{l=1}^{M_i} l \cdot z_i \cdot \frac{G_i-(K-l)}{G_{i-1}} \quad \text{for} \quad i = 1, \ldots, N \quad (13) \]

Note that the throughput \( \lambda_i \) cannot be computed by the following classical formula for infinite capacity networks [18]:

\[ \lambda_i(K) = \sum_{i=1}^{M_i} \mu_i \cdot p_i(n) \quad \text{for} \quad i = 1, \ldots, N \quad (14) \]

Here we introduce the following formula which provides the throughput of each station in the network.

\[ \lambda_i(K) = \frac{z_i}{G(K)} \left[ \sum_{j=1 \neq i}^{N} p_{ij} \cdot H_{ij}(K-1) \right] \quad \text{for} \quad i = 1, \ldots, N \quad (15) \]

where
\[ H_{ij}(K) = b_i(K) b_j(K) \prod_{t \neq i,j} a_t(K) \]  

(16)

with initial values

\[ H_{ij}(0) = 1 \]

The vectors \( a_i(K) \) and \( b_i(K) \) are defined as follows:

\[ a_i(K) = [1, z_i^1, z_i^2, \ldots, z_i^M, 0, 0, \ldots, 0] \]  

(17)

\[ b_i(K) = [1, z_i^1, z_i^2, \ldots, z_i^{M-1}, 0, 0, \ldots, 0] \]  

(18)

Other performance measures such as mean residence times \( \bar{\tau} \) and utilization \( \rho_i \) can be computed by Little's law and by classical utilization formula \( \rho_i = \lambda_i / \mu_i \), respectively.

6. Exact Analysis of Nonreversible Queueing Networks

We use the so-called duality concept where we show that there exists a dual queueing network to the originally given network. The dual queueing network contains "holes" as jobs which move to the opposite direction than in the originally given network model. A "hole" is the number of available places in the blocking network. The duality concept was first proposed by Gordon/Newell [8] for closed queueing networks with serially connected stations and service blocking where the blocked job stays at the head of the queue and resides there until a space becomes available in the destination station. Our solution is valid for rejection blocking queueing networks with serially as well as arbitrarily connected stations.

We denote the dual network parameters by plus signs in order to distinguish them from the parameters of the originally given network model. The parameters of the dual network are computed as follows. The service rates are determined by:

\[ \mu_i^+ = \sum_{j=1}^{N} p_{ji} \mu_j \]  

(19)

This equation is derived from the property that if a Poisson stream with rate \( \lambda \) thins (i.e., splits), i.e., jobs are independently allowed to continue with probability \( p \), the resulting stream is Poisson with rate \( p \lambda \).

Transition probabilities in the dual network are computed by:
This equation is derived from the property that if two Poisson streams with rates \(\lambda_1\) and \(\lambda_2\) are combined, the result is a Poisson stream with rate \(\lambda = \lambda_1 + \lambda_2\). Moreover, the conditional probability at an arbitrary instant that the next jump came from stream \(i\) given that a jump occurred is simply \(\lambda_i / \lambda\).

Now we state the following theorem.

**Theorem.** If in a closed network the total number of jobs is such that at most one station can be empty at a time, all other stations being full (i.e., \(M - K \leq M_i\) for all \(i\)) then the equilibrium state probabilities of the given blocking network have a product form solution.

**Proof.**

The dual queueing network has the same number of stations as the blocking queueing network \(N^* = N\). The difference is that the station capacities are unlimited, hence no blocking occurs and the total number of "holes" in the dual network \(K^* \neq K\). Another difference is that the "holes" in the dual network move in the opposite direction than in the blocking queueing network as mentioned above. The service times in the dual network are also exponentially distributed with rates \(\mu_i^*\), computed by equation (19) and the transition probabilities \(p_{ij}^*\) are determined by equation (20).

The behavior of the blocking network can be modeled by a Markov process \(X(t)\). The transition structure of \(X(t)\) can be described by the global balance equation:

\[
\sum_{i=1}^{N} \sum_{j=1}^{N} \mu_i p_{ij} \epsilon_i(k_i) \delta_j(k_j) p(k) = \\
\sum_{i=1}^{N} \sum_{j=1}^{N} \mu_j p_{ji} \epsilon_j(k_j) \delta_i(k_i) p(k_1, ..., k_i + 1, ..., k_I - 1, ..., k_N)
\]

where the binary functions \(\epsilon\) and \(\delta\) express the impossibility of jobs departing from a station that is empty and entering a station that is full:

\[
\epsilon_i(k_i) = \begin{cases} 
0 & \text{if} \quad k_i = 0 \\
1 & \text{if} \quad k_i > 0
\end{cases}
\]
\( \xi_{i}(k_{i}) \) is given previously.

The Markov process \( X(t) \) has the following state space:

\[
S = \{ k_{i} \mid \forall i \ (0 \leq k_{i} \leq M_{i}) \ \& \ \sum_{i=1}^{N} k_{i} = K \}
\]

Similarly, we define a Markov process \( X^{*}(t) \) whose transition structure is described by the following global balance equation for the dual network:

\[
\left( \sum_{i=1}^{N} \sum_{j=1}^{N} \mu_{i}^{+} p_{ij}^{+} \xi_{ij}(k_{i}^{+}) \right) p^{+}(k_{i}^{+}) = \\
\sum_{i=1}^{N} \sum_{j=1}^{N} \mu_{i}^{+} p_{ij}^{+} \xi_{ij}(k_{i}^{+}) p^{+}(k_{i}^{+}, ..., k_{i}^{+} + 1, \cdots, k_{j}^{+} - 1, \cdots, k_{N}^{+})
\]

The Markov process \( X^{*}(t) \) has the following state space:

\[
S^{*} = \{ k_{i}^{+} \mid \forall i \ (0 \leq k_{i}^{+} \leq K^{*}) \ \& \ \sum_{i=1}^{N} k_{i}^{+} = K^{*} \}
\]

We assert that \( S = S^{*} \)

i) The number of jobs in the dual network is defined in the following range:

\[
0 \leq k_{i}^{+} \leq K^{*}
\]

Replacing the values for \( K^{+} \) we get

\[
0 \leq k_{i}^{+} \leq \sum_{i=1}^{N} M_{i} - K
\]

Substituting \( k_{i}^{+} = (M_{i} - k_{i}) \) and considering \( K = \sum_{i=1}^{N} k_{i} \) we get

\[
0 \leq (M_{i} - k_{i}) \leq \sum_{i=1}^{N} M_{i} - \sum_{i=1}^{N} k_{i}
\]

Rewriting

\[
0 \leq \sum_{i=1}^{N} k_{i} - k_{i} \leq \sum_{i=1}^{N} M_{i} - M_{i}
\]

we obtain

\[
0 \leq \sum_{i=1}^{N} k_{i} \leq \sum_{i=1}^{N} M_{i}
\]

which provides
\[ 0 \leq k_i \leq M_i \]

ii) Substituting the value of \( K^+ \) we obtain

\[ \sum_{i=1}^{N} k_i^+ = \sum_{i=1}^{N} M_i - K \]

Rewriting

\[ \sum_{i=1}^{N} (M_i - k_i^+) = K \]

and substituting \( k_i = (M_i - k_i^+) \) we get

\[ \sum_{i=1}^{N} k_i = K \]

This implies that the equilibrium state probability \( p(k) \) of the blocking network is equivalent to the equilibrium state probability \( p^+(M - k) \) of the dual network:

\[ p(k) = p^+(M - k) \]

Substituting the value \( k^+ = (M - k) \) we obtain

\[ p(k) = p^+(M - k) \]

Since the dual network has product form solution, the \( p^+(M - k) \) values are obtained from the Gordon/Newell Theorem [7].

**Lemma.** The throughput of station \( i \) in the given closed queueing network with rejection blocking is equal to the throughput of station \( i \) in the dual network:

\[ \lambda_i(K) = \lambda_i^+(K^+) \quad \text{for} \quad i = 1, \ldots, N \] (27)

**Proof.**

Each time a job leaves station \( i \) to go to station \( j \), a hole leaves station \( j \) to go to station \( i \). As in equilibrium the arrival rate at a station has to be equal to its departure rate, the lemma follows.

We can compute the mean number of jobs \( \bar{E}_i^+ \) for the dual queueing network using a product form network algorithm. By taking the expected value of the definition of \( k_i^+ \) we get:

\[ \bar{E}_i = M_i - \bar{E}_i^+ \quad \text{for} \quad i = 1, \ldots, N \] (28)
Using Little's law we obtain the mean residence times as $\bar{\tau} = \bar{E}_t / \lambda_t$.

7. Conclusion

We have presented algorithms for computation of the normalization constant and formulas for other performance measures in queueing networks with rejection blocking and reversibility. This permits efficient analysis of such networks. As generally known the set of reversible networks is contained in the set of non-reversible networks. However, the exact analysis of non-reversible networks is possible under the condition that at most one station is allowed to be empty, which weakens the set of the non-reversible networks. We are in the process of finding an approximate solution for cases where the above mentioned condition, not satisfied. It is also interesting to investigate the cases where the stations have generally distributed service times and FCFS scheduling disciplines.

References


NETWORKS WITH MIXED PROCESSOR
SHARING PARALLEL QUEUES AND
COMMON POOLS

N. van Dijk* and I. F. Akyildiz**


* Vrije University
  Amsterdam
  Netherlands

** School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
U. S. A.

June 21, 1988
Networks with Mixed Processor Sharing Parallel Queues and Common Pools

N. M. van Do* and I. F. Akyildiz**

* Vrije University
Amsterdam
Netherlands

** School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

ABSTRACT

Networks of mixed exponential and non-exponential parallel queues with interdependent service capacities and common pools or accessibility constraints are studied. An invariance condition is provided in terms of the service and blocking protocols. It is shown that this condition guarantees a product form for the stationary joint queue size distribution. This distribution has the insensitivity property, i.e., it depends upon the service requirements only throughout their means. Various non-standard examples and applications are included. For instance, networks of processor sharing FCFS queues allowing different mean services for different job types, networks with reversible routing between parallel queues with common pools and networks with only a partial reversible routing and blocking

Key Words: Product Form, Insensitivity, Parallel Queues, Common Pools, Blocking

June 21, 1988

Akyildiz's work was supported in part by School of Information and Computer Science, ICS, of Georgia Tech and by the Air Force Office of the Scientific Research (AFOSR) under Grant AFOSR-88-028.
Networks with Mixed Processor Sharing Parallel Queues and Common Pools

N. M. van Dijk* and I. F. Akyildiz**

* Vrije University
Amsterdam
Netherlands

** School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
U. S. A.

1. Introduction

A queueing network is a collection of stations in which jobs proceed from one to another to satisfy their service requirements. Queueing networks have enjoyed increasing popularity over the last two decades as models for telecommunication networks, computer systems, and flexible manufacturing systems. These applications frequently feature situations in which a number of parallel queues have to share a common pool or are attached to some joint processor.

Product form results for queueing networks and their relationships with notions of partial balance have been extensively studied over the last two decades [4, 6, 7, 8, 12, 14, 17, 23, 34]. For exponential networks with a fixed routing, no blocking and station independent servicing the product form is a common feature. For networks with blocking or load dependent servicing the results are much more restrictive. The product form is generally restricted for exponential networks with finite queue size constraints provided the routing is completely reversible while the servicing is load-independent [1, 2, 11, 17, 26]. Although blocking results with nonreversible routing have been reported [11, 12] various situations with capacity constraints remain open such as with a routing which is only partially reversible. Conversely, for exponential networks with fixed routing and no blocking, product form results have also been reported with station interdependent servicing provided the service rates at a particular queue are defined by a special functional form [7, 17, 33].

These product form results for exponential networks, remain valid for networks with non-
exponential queues (insensitivity phenomenon) provided at these queues a detailed notion of partial balance is satisfied per position or per job (local or job-local balance). This notion is guaranteed when the service discipline is symmetric \([3,7,8,17]\) or satisfies a more general service invariance condition \([11]\).

The network model under consideration has not been covered by the above references as

i) it includes a blocking due to common constraints of collections of parallel queues,

ii) it allows service interdependencies within such collections and

iii) it does not require reversibility of the routing all over but only where blocking can occur.

The literature on systems with parallel queues is rather extensive due to their practical interest but has been restricted to commonly shared pools with assumptions of Poisson inputs and exponential services \([9,15,19,20,29,30]\). Under these assumptions product form results have been established. In practice, however, exponential services are rather restrictive and input stream such as in closed systems are usually non-Poisson.

This paper aims to extend the above results to non-exponential services and no Poisson input requirements and interdependencies of parallel queues both in rejecting and servicing jobs. The main results of this paper are:

i) An insensitive product form expression

ii) A concrete blocking and service invariance condition

iii) A number of new product form examples with the novel aspects of:

a) A general interdependent blocking of parallel queues

b) A general interdependent servicing of parallel queues

c) A reversible routing only where blocking can arise

The proof of the product form result is straightforward and self-contained as based upon verifying particular balances. The insensitivity is concluded by the notion of balance per job and an intermediate step with mixtures of Erlang distributions. Although it is a standard result, this latter step is included as it does not require essentially more work while it makes the proof self-contained. The presentation is restricted to closed queueing networks. However, the extension to open queueing networks is
The organization of this paper is as follows: Section 2 describes the various model protocols. The essential invariance condition on the blocking and servicing protocols is presented in section 3. The product form result is derived in section 4. Several examples which illustrate the invariance condition on the blocking and service protocols are given in section 5. Evaluation and a list of symbols concludes the paper.

2. Network Description

The system consists of $N$ stations having multiple servers and $M$ fixed number of jobs. There are $T$ possible job types where $T$ is allowed to be infinite. Each station $s$ has $Q(s)$ parallel queues, for $s = 1, ..., N$. A job entering station $s$ requires service at one of these queues depending on the present type number of the jobs, $g(s,t)$. After completing service a job of type $t$ at queue $q(s,t)$ of station $s$ goes with probability $p_{s,t}^{s'}$ to the queue $q'(s',t')$ of the station $s'$ and changes its type to $t'$. For later convenience we assume without loss of generality that $p_{s,t}^{s'} = 0$ for all $s,s',t,t'$. The job can be rejected by the destination station $s'$ based upon the present job configuration at that station. This blocking and its protocol will be described in section 2.1. Various queues at a station provide service at interdependent service rates as will be described in section 2.2. The service allocation to the jobs at a queue is governed by a queueing discipline which will be described in section 2.3. In section 2.4 we specify the service distributions.
2.1. Blocking Protocol

Let $\bar{n}_s = (n^1_s, n^2_s, \ldots, n^T_s)$ denote that $n^t_s$ jobs of type $t$ are present at station $s$, for $t = 1, 2, \ldots, T$ and $s = 1, \ldots, N$. Suppose that a job of a type $t'$ completes service at station $s'$ and requests service at station $s$ with its type number changed into $t$, while the current configuration of the jobs already present at station $s$ is given by $\bar{n}_s$. This request is accepted with probability

$$A_s(q' | \bar{n}_s)$$

and the job is allocated to queue $q$ at station $s$. If the request is rejected, the job retains its type number $t'$ and has to restart a new service at station $s'$ as a new arriving job. Note that the rejected job has to be accepted by the source station.

This blocking protocol is known as the rejection [1, 2] or Type III [24] or also as communication blocking [17,26].
One may observe that the function $A_s(q(s,t) \mid \bar{n}_s)$ allows the blocking probability of a type $t$ job to depend on not only the total number of type $t$ jobs (such as due to a capacity constraint at the corresponding queue $q(s,t)$ but also on the number of jobs of other types (such as due to a common in-or output channel). This multi-type dependent blocking will be restricted by a general invariance condition in section 3.

2.2. Service Rates

If station $s$ is in state $\bar{n}_s = (n_s^1, n_s^2, \ldots, n_s^T)$ denoting that $n_s^t$ jobs of type $t$ are present at one of its servers $t = 1, 2, \ldots$, then the number of jobs at each individual queue is given as each type-$t$ job has a corresponding queue number $q(s,t)$. (Note that more job types may be allocated to the same queue).

The rate at which queue $q$ provides service can be specified by

$$
\Phi_s(q \mid \bar{n})
$$

where we assume that this function has either the value 0 if there are no jobs at queue $q$ or has positive value otherwise.

Note that by this definition we allow the rate out of one queue to depend on the number of jobs at other servers (such as due to a common acceleration if the total number of jobs at station $s$ exceeds some threshold). The server dependency will be restricted by a general invariance condition for blocking protocol in section 3.

2.3. Service Disciplines

Consider a queue $q$ at station $s$ while the job configuration at the station is given by $\bar{n}_s = (n_s^1, n_s^2, \ldots, n_s^T)$. Let $x_p$ be the number of jobs at this queue and note that

$$
\Phi_s(q \mid \bar{n})
$$

is the total amount of service provided at this queue per unit of time.

Then the service allocation to the individual jobs at this queue is governed by positions as follows:

The $x_p$ jobs are positioned at $1, \ldots, x_p$. Then
\( \Gamma_{s,q}(p | \pi) \) is the fraction of the total amount \( \Phi_{s}(q | \pi) \) assigned to the job at the \( s \)-th position, \( p = 1, \ldots, z_p \).

\( \delta_{s,q}(p | \pi) \) is the probability that the last entered job at queue \( q \) from the jobs present has been assigned position \( p, p = 1, \ldots, z_p \).

When a job at position \( p \) completes its service the jobs at positions \( p+1, \ldots, z_p \) shift to positions \( p, \ldots, z_p-1 \). When a job is assigned position \( p \), the jobs at previous positions \( p, \ldots, z_p \) are shifted to positions \( p+1, \ldots, z_p+1 \). We assume hereby that

\[
\sum_p \Gamma_{s,q}(p | \pi) = \sum_p \delta_{s,q}(p | \pi) = 1 \tag{1}
\]

We will distinguish two types of service disciplines. A discipline is said to be non-symmetric when it adopts the above description without further conditions. A discipline is said to be symmetric when in addition

\[
\Gamma_{s,q}(p | \pi) = \delta_{s,q}(p | \pi) \quad p = 1, \ldots, z_p + 1 \quad \text{for all } n \tag{2}
\]

Let \( S \) be the set of all symmetric queues. The term (non)-symmetric corresponds to the definition in Kelly [17]. Various practical disciplines can be parametrised in the above manner [17]. Most notably are the standard BCMP disciplines [4]:

- FCFS: First Come First Served (\( \notin S \))
- PS-1: Processor Sharing single server (\( \in S \))
- IS: Infinite Servers (\( \in S \))
- LCFS-PR: Last Come First Served Pre-emptive Resume (\( \in S \))

2.4. Distribution Functions

The service distribution of a job of class \( t \) at station \( s \) depends upon the service discipline of its queue \( q(s,t) \) and has a distribution function of the form.

\[
G_s^t = \begin{cases} E(1, \mu_{s,t}) & \text{for } q(s,t) \notin S \\ \sum_{k=1}^{\infty} a^t_k(k) E(k, v') & \text{for } q(s,t) \in S \end{cases} \tag{3}
\]

where \( E(k, \alpha) \) denotes an Erlang \( k \)-distribution with mean \( k/\alpha \) and where \( a^t_k(k) \) denotes the probability that the distribution consists of \( k \) successive exponential phases with parameter \( \nu' \) assuming
\[ \sum_{k} a_t^i = 1. \text{ Hence,} \]

\[ \tau_i^t = \begin{cases} 
\frac{1}{\mu_i} & \text{for } q(s,t) \notin S \\
\sum_{k=1}^{\infty} a_t^i(k) E(k,\nu_i) & \text{for } q(s,t) \in S 
\end{cases} \quad (4) \]

is the mean service requirement of a type \( t \)-job at station \( s \) while

\[ R_i^t(r) = \frac{[\sum_{k=1}^{\infty} a_t^i(k)]^r}{[\nu_i^t \tau_i^t]} \quad (5) \]

is known from the renewal theory [18] as the stationary excess probability of \( \tau^r \) residual exponential phases up to a next renewal in a renewal process with renewal function \( G_i^t \) for \( q(s,t) \in S \). Informally, the function (3) requires all jobs at a non-symmetric queue to have an exponential service with one and the same parameter regardless of job type, while a job at a symmetric queue may have a general mixture of Erlang service distributions depending on its job type. The restriction to these mixtures will be used in section 4 to justify a Markovian analysis. The proof of our results will thus be established for these mixtures only. It is well-known, however, that any nonnegative probability distribution can be arbitrarily closely approximated by these mixtures (in the sense of weak convergence, [10]). Based upon standard weak convergence limit theorems for the probability measures of the sample paths on appropriate so-called \( D \)-spaces [3,13,85], the insensitivity result can therefore be extended to arbitrary service distributions.

3. Conditions

First it is to be noticed that the routing probabilities \( p_{\cdot \cdot} \), the possible changes of job types and the blocking functions \( A(\cdot,\cdot) \) together with the blocking protocol will exclude certain configurations which are given below. Let \( R \) be a set of all reachable configurations \( N = (\overline{n}_1, \ldots, \overline{n}_N) \) with a given starting configuration \( N^o = (\overline{n}_1^o, \ldots, \overline{n}_N^o) \) and exponential sojourn times with unit mean at any queue for any job. We assume \( R \) to be irreducible. Throughout of this paper, we will restrict our attention to configurations within \( R \). We define a station configuration \( \overline{n}_s \) admissible if there exists a configuration.
within $R$ with $\pi_s$ restricted to station $s$. We are now ready to present our conditions.

3.1. Partial Reversible Routing

The routing probabilities from one station to another are subject to a partial reversibility condition. Informally, it requires the routing to be reversible wherever jobs can be rejected, but it allows arbitrary fixed routing probabilities where jobs cannot be rejected. More precisely, without loss of generality, assume that there exists a unique probability distribution

$$\{ \lambda_s^t : s = 1, \ldots, N, t \in \{1, 2, \ldots, T\} \}$$

satisfying the traffic equations

$$\lambda_s^t = \sum_{s', t'} \lambda_{s'}^{t'} p_{s's}^{t' t} \quad (s = 1, \ldots, N ; t \in T) \tag{8}$$

Then additionally we require the following Partial Reversibility Condition:

For any station $s$ and type $t$ such that for some admissible configuration $\pi_s$ and queue $q = q(s, t)$:

$$A_s(q \mid \pi_s) < 1 \tag{7}$$

we have

$$\lambda_s^t p_{s's}^{t' t} = \lambda_{s'}^{t'} p_{s's}^{t' t} \quad \text{for all } s', t' \quad \text{with } p_{s's}^{t' t} > 0 \text{ or } p_{s's}^{t' t} > 0 \tag{8}$$

Note that the standard reversibility condition [17] requires equation (8) to hold for any $(s, s', t, t')$. While in our case it is required only for a subset satisfying equation (7). Note also that our partial reversibility condition, equation (8) has merely to do with routing in contrast to the quasi reversibility [17].

The reason for including this partial rather than global reversibility condition is twofold:

i) It allows us to simultaneously analyse systems with as well as without blocking. Even without blocking the results of this paper are new as they involve a service interdependence of parallel queues.

ii) New examples with blocking but a global non-reversible structure can be covered. Blocking results in the literature either require a reversibility all over the network [1, 2, 13, 17, 26] or provide a general non-reversibility routing condition but exclude for instance first come first served queues [11]. As an example we give the following non-reversible structure where we assume only one-job
type so that we can delete job type specification and consider stations with only one queue ($q = 1$):

Transition probabilities are:

$$P_{12} = P_{22} = P_{32} = P_{21} = P_{14} = \frac{1}{2};$$

$$P_{ij} = 1 \quad \text{otherwise}$$

and

$$A_2(1 \mid \bar{n}_2) = \begin{cases} 1 & \text{for } n_2 < N_2 \\ 0 & \text{for } n_2 = N_2 \end{cases}$$

satisfies (6, 7 and 8) with

$$\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{6}$$

$$\lambda_f = \frac{1}{12} \quad \text{for all } j \neq 1, 2, 3$$

while it allows a finite capacity constraint at the central station 2.
3.2. Blocking and Service Invariance Condition

In order to present a general condition upon the interdependent blocking and servicing at a particular station we need to introduce some notation. We will focus on a fixed station \( s \). For a vector \( \bar{\pi} = (n^1, n^2, \ldots, n^T) \) with \( n^1 + n^2 + \cdots + n^T = \pi \) denoting that \( \pi \) jobs are present at station \( s \) of which \( n^t \) are of type \( t, t \in \{1, 2, \ldots, T\} \). Let \( T(\bar{\pi}) \) be the corresponding vector with the first \( n^1 \) components equal to \( t_1 \), the first type number \( t \) in increasing order with \( n^t > 0 \), the next \( n^{t_2} \) equal to \( t_2 \), the second \( t \) with \( n^t > 0 \), etc. Conversely, for any given vector of type numbers \( (j_1, j_2, \ldots, j_k) \) let \( \bar{\pi}(j_1, \ldots, j_k) \) be the vector of corresponding numbers \( n_i \) of jobs of type \( t \in \{1, 2, \ldots, k\} \). Furthermore throughout for a given number \( \bar{\pi} \) at station \( s \), let \( \bar{m}_s \equiv (m_1^s, \ldots, m_{Q(s)}^s) \) denote the corresponding queue sizes \( m_j \) at queue \( q=1, \ldots, Q(s) \) and let \( m_s = m_1^s + \cdots + m_{Q(s)}^s = \pi \) be the total number at station \( s \).

**Invariance Condition:**

For any station \( s \), any admissible vector \( \bar{\pi}_s \) (at station \( s \)) and with \( T(\bar{\pi}_s) \) the corresponding queue size vector of size \( \pi \), the product

\[
\Pi_{j=1}^{\pi} \frac{A(s, j) | \bar{\pi}(j_1, \ldots, j_{\pi-1})}{\Phi_s(s, j) | \bar{\pi}(j_1, \ldots, j_{\pi-1}, j_{\pi})}
\]

is invariant for all permutations

\[
(j_1, \ldots, j_{\pi}) \in T(\bar{\pi}_s)
\]

This invariance product is denoted by

\[
P_s(\bar{\pi}_s)
\]

while we introduce

\[
P_s(\bar{\pi})=1 \quad \text{for} \quad \pi = 0
\]

Informally, this condition requires that it does not matter in which order the jobs of the various types arrive if we consider \( \frac{A(\cdot, \cdot)}{\Phi(\cdot, \cdot)} \) as state dependent arrival rate.
REMARK.

i) Note that as we have required $\Phi(q | \bar{F}) > 0$ for any queue $q$ and $\bar{F}$, the functions $A_s(\cdot | \bar{F})$ must necessarily be positive for reaching any admissible vector $\bar{F}_s$. We will have, however, $A_s(\cdot | \bar{F}) = 0$ when the acceptance of a next job would lead to a non-admissible state vector.

ii) (Decoupling). Clearly, the invariance condition is guaranteed by separately unifying the invariance of the products

$$\prod_{k=1}^{n} A_s(q(k), \pi) | \pi(j_1, \ldots, j_{n-1})$$

and

$$\prod_{k=1}^{n} \Phi_s(q(k), \pi) | \pi(j_1, \ldots, j_{n-1})$$

for all permutations $(j_1, \ldots, j_n) \in T(\bar{F}_s)$. Although examples can be found for which (9) holds while (11) and (12) fail, the conditions (11) and (12) seem much more realistic as they decouple blocking and servicing. We will therefore restrict our examples given in section 5 to (11) and (12).

iii) (Convex Blocking). An important subclass of blocking satisfying the invariance condition (11) is obtained by assuming that $A_s(\cdot | \bar{F}_s)$ depends upon $\bar{F}_s$ only by $\bar{m}_s$; the vector of queue sizes $m_1, m_2, \ldots$ by

$$A_s(q | \bar{F}_s) = \begin{cases} 1 & \text{if } \bar{m}_s + e^f \in B_s \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{m}_s + e^f$ denotes the vector equal to $\bar{m}_s$ with one job more (+ sign) or less (- sign) at queue $q$ and where $B_s$ is a set such that

$$\bar{m}_s = (m_1, \ldots, m_{Q(s)}) \in B_s$$

that is, blocking arises only to prohibit departures from $B_s$ where $B_s$ satisfies (14). Due to [9,16] we call such blocking coordinate convex. The verification of (11) is immediate as accessible states $\bar{F}_s$ are necessarily restricted to $B_s$ so that the product (11) is equal to 1 for any accessible state $\bar{F}_s$ regardless of the chosen permutation.
4. Product Form Solution

In this section we will derive the insensitive product form results. As different stations, queues, positions, job-types and residual service amounts need to be specified, some notational complexity is unavoidable. Let

$$\mathcal{V} = [\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_N]$$

with

$$\mathcal{V}_s = [\mathcal{Q}_1^s, \mathcal{Q}_2^s, \ldots, \mathcal{Q}_{Q(s)}^s] \quad \text{for} \quad (s = 1, \ldots, N)$$

where

$$\mathcal{Q}_q^s = [(t_{q,s}^s, r_{q,s}^s), \ldots, (t_{q,s}^{s+1}, r_{q,s}^{s+1})] \quad \text{for} \quad (q = 1, \ldots, Q(s))$$

to denote for each station $s$ and each queue $q$ at this station that $m_{q,s}$ jobs are present at this queue of which the job at position $p$ has a job-type number $t_{q,s}^p$ and a residual number of exponential service phases $r_{q,s}^p$. Further, for a given state $\mathcal{V}$ let

$$\mathcal{V} + [s', q'] (t', p', r') - [s, q] (t, p, r)$$

be the state that differs from $\mathcal{V}$ in that the job at position $p$ in queue $q$ at station $s$ with $t_{q,s}^p = t$ and $r_{q,s}^p = r$ has moved to position $p'$ in queue $q'$ at station $s'$ with $t_{q',s'}^{p'} = t'$ and $r_{q',s'}^{p'} = r'$. Here it is to be noted that for a job at a non-symmetric queue the number of residual exponential service phases is necessarily equal to 1. Similarly, for a given vector $\vec{n}_s = (n_s^1, n_s^2, \ldots, n_s^T)$. Let

$$\vec{n}_s + e_t^t$$

denote the vector that differs from $\vec{n}_s$ in one job more (+ sign) or less (- sign) from type $t$. Finally, recall that $q(s, t)$ is the queue number for a job type $t$ at station $s$ and that $S$ denotes the set of all symmetric queues.

Now we are ready to present the two main theorems of this paper. The first theorem is the key theorem which contains detailed information than needed. The second theorem is more practical consequence of theorem 1 and shows the insensitivity property. Now let us assume that there exists a unique stationary distribution $\Pi(.)$ for the $\mathcal{V}$ process restricted to an irreducible set $\mathcal{R}$. 
Theorem 1. With $C$ as a normalization constant and $P_s(.)$ defined by (9) we have

$$
\Pi(V) = C \prod_{i=1}^{N} P_s(\pi_i) \left\{ \prod_{t=1}^{T} (\lambda_j)^{n_t} \right\} \left\{ \prod_{t \in s} \left( \frac{1}{\mu_{s,t}} \right)^{n_{s,t}} \right\} \left\{ \prod_{s \in S} \prod_{r \in S} r_{s,t}^{n_{s,t}} R_{s,t}^{n_{s,t}} \right\} \tag{15}
$$

Before presenting the proof, let us give a direct consequence of this theorem. Now let

$$\bar{N} = (\bar{\pi}_1, \bar{\pi}_2, ..., \bar{\pi}_N)
$$

where

$$\bar{\pi}_s = (n_1, n_2, ..., n_r)
$$

to denote that $n_t$ jobs of type $t$ are present at station $s$ for all possible $t$ and $s$. By standard calculus or from renewal theory [18] notice that for any $s, t$:

$$\sum_{r=0}^{\infty} R_t^s(r) = 1 \tag{16}
$$

Therefore by summing over all possible numbers $r = r_{s,t}$ of residual exponential phases for any $p, s, q$ and by disregarding the specification of positions $p$ for the jobs individually, we can conclude the following main result from theorem 1. It shows that the steady state joint queue size vector has a product form and is insensitive for symmetric queues.

Theorem 2. With $C$ as a normalization constant, the steady state distribution for admissible states is given by:

$$
\Pi(\bar{N}) = C \prod_{i=1}^{N} P_s(\bar{\pi}_i) \left\{ \prod_{t=1}^{T} (\lambda_j)^{n_t} \right\} \left\{ \prod_{t \in s} \left( \frac{1}{\mu_{s,t}} \right)^{n_{s,t}} \right\} \left\{ \prod_{t \in s, t \in S} (r_t)^{n_t} \right\} \tag{17}
$$

In the following we present the proof of theorem 1.

Proof.

By virtue of the Markovian structure of the $V$ process it is sufficient to verify the global balance (or equilibrium) equations, [18, pp. 92]. These require that the total rate (or probability flow) out of any state due to a change at any of the queues $q = 1, ..., Q(s)$ for $s = 1, ..., N$, is equal to the rate into that state due to a change at any of these queues. However, this in turn is guaranteed if for each queue $q \in \{1, ..., Q(s)\}$ and for $s = 1, ..., N$ individually, we can establish:
The rate out of any state due to a change at queue \(q\) is
\[
= \text{The rate into that state due to a change at queue } q. \quad (18)
\]

In the following we will verify (18) for non-symmetric and symmetric queues, respectively. In particular, for symmetric queues we will even establish (18) by

The rate out of any state due to a change at position \(p\) at queue \(q\) is
\[
= \text{The rate into that state due to a change at position } p \text{ at queue } q. \quad (19)
\]

Now consider a fixed state \(\mathcal{V}\) and station \(s \in \{1,\ldots,N\}\)

i) **Non-symmetric Queue.** Consider a fixed queue \(q \in \{1,\ldots,Q(s)\}\) and for convenience let

\[
t(\rho) = c'_\rho \quad \text{for } \rho = 1,\ldots,n_s.
\]

The rate out of state \(\mathcal{V}\) due to a change at queue \(q\) is given by

\[
\Pi(\mathcal{V}) \cdot \Phi_s(q | \mathcal{W}_s) \cdot \mu_{s,q} \cdot \left[ \sum_p \Gamma_{s,q}(p | \mathcal{W}_s) \right] \quad (20)
\]

The rate into this state due to a change at \(q\) is equal to

\[
\sum_p \left\{ \sum_{n' \neq p} \sum_{n''} \Pi(\mathcal{V} + [s',q(s',\rho) | q',\rho']) (t(\rho'),p',1) - [s,q] (t(\rho),p,1)) \cdot \Phi_{s'}(q(s',\rho') | \mathcal{W}_s + c'_\rho) \cdot \Gamma_{s',q}(s',\rho') \cdot \nu_{s',p}^{n'} \cdot p_{s',p}^{n'}(s',\rho') \cdot \delta_{s',p}(p | \mathcal{W}_s) \cdot A_s(q | \mathcal{W}_s - c'_s) \right\}
\]

\[
\cdot \sum_p \left\{ \sum_{n''} \Pi(\mathcal{V} + [s,q] (t(\rho),p',1) - [s,q] (t(\rho),p,1)) \cdot \mu_{s,q} \cdot \Phi_{s}(q | \mathcal{W}_s) \cdot \Gamma_{s,q}(p' | \mathcal{W}_s) \cdot \left[ \sum_{n'} \nu_{s'}^{n'} \cdot p_{s',p}^{n'} (1 - A_{s'}(q(s',\rho') | \mathcal{W}_s)) \cdot \delta_{s',p}(p | \mathcal{W}_s) \right] \right\}
\]

where

\[
\nu_s^{n'} = \mu_{s,q}(s',\rho)
\]

when

\[
q(s',\rho') \notin S
\]

Now first recall that

\[
A_s(q | \mathcal{W}_s - c'_s) > 0
\]

for any admissible \(\mathcal{W}_s\) and \(q\) as we have assumed

\[
\Phi_s(q | \mathcal{W}_s) > 0
\]
provided \( n > 0 \).

Similarly

\[ \Phi_x(q(s', t') | \pi_\nu + e_\nu^x) > 0 \]

for any admissible \( \pi_\nu + e_\nu^x \).

As a result, by substituting (15) and using the invariance of the product (17) for expression (18) we conclude that for any admissible state \( V \) and admissible state \( \{ V + [s', q(s', t')] \} (t', p, 1) - [s', q](t(p), p, 1) \) with \( s' \neq s \).

\[
\Pi (V + [s', q(s', t')](t', p, 1) - [s', q](t(p), p, 1)) =
\]

\[
\Pi (V) \cdot \left[ \frac{A_x(q(s', t') | \pi_\nu)}{A_x(q | \pi_\nu - e_{\nu}^{(x)})} \right] \cdot \left[ \frac{\Phi_x(q | \pi_\nu)}{\Phi_x(q(s', t') | \pi_\nu + e_\nu^x)} \right] \cdot \left[ \frac{\lambda_\nu^{(x)}}{\lambda_\nu^{(s)}} \right] \{ 1 \cdot (q(s', t')) \cdot \mu_s \cdot \tau \cdot R_{s}^{(x)}(1) + \left[ 1 - 1 \cdot (q(s', t')) \right] \frac{\mu_s \cdot \tau \cdot R_{s}^{(x)}(1)}{\mu_{s'} \cdot \tau \cdot R_{s'}^{(x)}(1)} \}
\]

where

\[
1_s(q) = \begin{cases} 1 & \text{for } q \in S \\ 0 & \text{for } q \notin S \end{cases}
\]

Furthermore also by (15) it is valid that

\[
\Pi (V + [s, q](t(p), p', 1) - [s, q](t(p), p, 1)) = \Pi (V)
\]

for all \( p, p' \).

By substituting (22) and (24) into (21) and using

\[ \sum_{p'} \Gamma_{s', s}(p') = \sum_{p'} \Gamma_{s, s}(p') = 1 \]

as by (1) and recalling

\[ \nu_{s'}^{(x)} = \mu_{s'} \cdot \tau \cdot R_{s'}^{(x)}(1) \quad \text{for } q(s', t') \notin S \]

we can then rewrite (21) as

\[
\Pi (V) \cdot \Phi_x(q | \pi_\nu) \cdot \mu_s \cdot \sum_p \delta_{s, s}(p | \pi_\nu)
\]

(25)
\[
\left\{ \frac{1}{\lambda_s(p)} \cdot \left[ \sum_{s' \in S} p_{s',s}^{q,t}(p) \cdot \lambda_{s,s}^{q,t} \cdot A_{s'}(q(s',t') | \bar{m}_s) \cdot \nu_{s,s}^{q,t} \cdot \tau_{s,s}^{q,t} \cdot R_{s,s}^{q,t}(1) \right. \\
\left. + \sum_{s' \in S} p_{s',s}^{q,t}(p) \cdot \lambda_{s,s}^{q,t} \cdot A_{s'}(q(s',t') | \bar{m}_s) \cdot \nu_{s,s}^{q,t} \cdot \tau_{s,s}^{q,t} \cdot \{ 1 - A_{s'}(q(s',t') | \bar{m}_s) \} \right]\} 
\]

Noting that
\[
R_{s,s}^{q,t}(1) = \frac{1}{\nu_{s,s}^{q,t} \tau_{s,s}^{q,t}} 
\]
and by virtue of (4) we thus obtain for (21)
\[
\Pi(\mathcal{V}) \cdot \Phi_s(q | \bar{m}_s) \cdot \mu_{s,t} \cdot \sum_p \delta_{s,t}(p | \bar{m}_s) \cdot \\
\frac{1}{\lambda_s(p)} \cdot \left\{ \sum_{s' \in S} \lambda_{s,s}^{q,t} p_{s',s}^{q,t}(p) \cdot A_{s'}(q(s',t') | \bar{m}_s) + \sum_{s' \in S} \lambda_{s,s}^{q,t} p_{s',s}^{q,t}(p) \cdot \{ 1 - A_{s'}(q(s',t') | \bar{m}_s) \} \right\} 
\]

Now the partial reversibility conditions (7 and 8) need to be taken into account. When
\[
A_{s'}(q(s',t') | \bar{m}_s) = 1 \quad \text{for all } s', t'
\]
with \(p_{s',s}^{q,t}(p) > 0\) or \(p_{s',s}^{q,t}(p) > 0\), the second term within the bracket of (27) is equal to 0 and the equality of (20) and (27) and thus (21) directly follow from the traffic equations (6) and (1).

When however
\[
A_{s'}(q(s',t') | \bar{m}_s + e_{s'}^{q}) < 1 \quad \text{for some } s', t'
\]
with \(p_{s',s}^{q,t}(p) > 0\) or \(p_{s',s}^{q,t}(p) > 0\) and assuming \(\bar{m}_s + e_{s'}^{q}\) to be admissible, the partial reversibility conditions (7 and 8) reduce equation (27) to
\[
\Pi(\mathcal{V}) \cdot \Phi_s(q | \bar{m}_s) \cdot \mu_{s,t} \cdot \sum_p \delta_{s,t}(p | \bar{m}_s) \cdot \\
\frac{1}{\lambda_s(p)} \left\{ \sum_{s' \in S} \lambda_{s,s}^{q,t} p_{s',s}^{q,t}(p) \left[ A_{s'}(q(s',t') | \bar{m}_s) + \{ 1 - A_{s'}(q(s',t') | \bar{m}_s) \} \right] \right\} 
\]
so that also in this case equality of (20 and 21) is proven by the virtue of \(\sum_{s' \in S} p_{s',s}^{q,t} = 1\) and equation (1).

As the state \(\mathcal{V}\), station \(s\) and queue \(q\) of station \(s\) were arbitrarily chosen, we have hereby
verified (18) for any non-symmetric queue \( q \).

ii) Symmetric Queue. Now consider a fixed queue \( q \in \{1, \ldots, Q(s)\} \) as well as a fixed position \( p \in \{1, \ldots, n_q\} \) at this queue. For convenience let \( t = t^*_{q,p} \) and \( r = r^*_{q,p} \). In the following we verify (18) through (19).

The rate out of state \( V \) due to a change at position \( p \) of queue \( q \) is given by

\[
\Pi(V) \cdot \Phi_s(q | \overline{q}) \cdot \Gamma_{s,q}(p | \overline{q}) \cdot \nu^*_{s}
\]

(29)

The rate into state \( V \) due to a change at position \( p \) of queue \( q \) equals

\[
\Pi(V) \left[ \Phi_s(q | \overline{q}) \cdot \Gamma_{s,q}(p | \overline{q}) \cdot \nu^*_{s} + \sum_{r', \bar{p}} \sum_r \Pi(V) \left[ \Phi_s(q' | \overline{q} + \varepsilon) \cdot \Gamma_{s,q}(p' | \overline{q} + \varepsilon) \cdot \Phi_{s'}(q' | \overline{q} + \varepsilon) \cdot \delta_{s,s}(p | \overline{q}) \right] \cdot \rho_{s,s}(r) + \right]
\]

\[
\sum_{s', \bar{p}} \Pi(V) \left[ \Phi_s(q' | \overline{q}) \cdot \Gamma_{s,q}(p' | \overline{q}) \cdot \nu^*_{s'} \cdot \sum_{r'} \rho_{s,s'}(r') \cdot \left[ 1 - A_{s'}(q(s', \ell') | \overline{q}) \right] \cdot \delta_{s,s}(p | \overline{q}) \cdot \rho_{s,s}(r) \right]
\]

(30)

where as before \( \nu^*_{s'} = \mu_{s,s'}(\ell') \) for \( q(s', \ell') \notin S \).

By taking the remark made after equation (21) and also equation (23) into account we conclude similarly to (22) and (24) that:

\[
\Pi(V) \left[ \Phi_s(q | \overline{q}) \cdot \Gamma_{s,q}(p | \overline{q}) \cdot \nu^*_{s} \right] = \Pi(V) \left[ \frac{R'_s(r+1)}{R'_s(r)} \right]
\]

(31)

\[
\Pi(V) \left[ \Phi_s(q | \overline{q}) \cdot \Gamma_{s,q}(p | \overline{q}) \cdot \nu^*_{s} \right] = \Pi(V) \left[ \frac{R'_s(1)}{R'_s(r)} \right]
\]

(32)

\[
\Pi(V) \left[ \Phi_s(q | \overline{q}) \cdot \Gamma_{s,q}(p | \overline{q}) \cdot \nu^*_{s} \right] = \Pi(V) \left[ \frac{\lambda'_s}{\lambda'_s} \right]
\]

(33)
By substituting (31), (32) and (33) into (30) and using

$$
\sum_{p'} \Gamma_{p',p}(p'|.) = \sum_{p'} \Gamma_{p,p}(p'|.) = 1
$$

as by (1) and recalling

$$
\nu_{s}^{\phi} = \mu_{s,p}(s'|.) \quad \text{for} \quad q(s',t') \notin S
$$

Similarly to (24) we can rewrite (30) as:

$$
\Pi(\mathcal{V}) \cdot \Phi_{s}(q | \mathcal{R}) \cdot \Gamma_{s,p}(p | \mathcal{R}) \cdot \frac{\nu_{s}^{t}}{[R_{s}'(r)]} \cdot \left[ R_{s}'(r+1) + \left[ \frac{\delta_{s,p}(p | \mathcal{R})}{\Gamma_{s,p}(p | \mathcal{R})} \cdot \frac{a_{s}'(r)}{[\nu_{s}' \cdot \tau_{s}']} \cdot \left[ \sum_{p' \in S} \rho_{s}^{p'} \cdot \lambda_{s}^{p'} \cdot A_{s}(q(s',t') | \mathcal{R}) \right] \right] \right.
$$

$$
\cdot \sum_{p' \in S} \rho_{s}^{p'} \cdot \lambda_{s}^{p'} \cdot A_{s}(q(s',t') | \mathcal{R}) \cdot \nu_{s}^{t} \cdot \tau_{s}^{t} \cdot R_{s}'(1) + \sum_{p' \in S} \lambda_{s}^{p'} \cdot \rho_{s}^{p'} \cdot \left[ 1 - A_{s}(q(s',t') | \mathcal{R}) \right] \frac{\nu_{s}^{t} \cdot \tau_{s}^{t} \cdot R_{s}'(1)}{\lambda_{s}^{t}} \}
$$

The symmetry condition (2) and expression (26) reduce this expression (34) to:

$$
\Pi(\mathcal{V}) \cdot \Phi_{s}(q | \mathcal{R}) \cdot \Gamma_{s,p}(p | \mathcal{R}) \cdot \frac{\nu_{s}^{t}}{R_{s}'(r)} \cdot \left[ \sum_{p' \in S} \lambda_{s}^{p'} \cdot \rho_{s}^{p'} \cdot A_{s}(q(s',t') | \mathcal{R}) \sum_{p'} \lambda_{s}^{p'} \cdot \rho_{s}^{p'} \cdot \left( 1 - A_{s}(q(s',t') | \mathcal{R}) \right) \right]
$$

with

$$
R_{s}'(r) = R_{s}'(r+1) + \frac{a_{s}'(r)}{[\nu_{s}' \cdot \tau_{s}']}
$$

According to (5), equality of (29) and (35) (and thus (30)) now follows similarly to that of (20) and (27) as based upon the traffic equation (6) and the partial reversibility condition (7 and 8). We have thus verified (19) for any \( p \) so that (18) is also secured. With \( \mathcal{V} \) station \( s \) and queue \( q \) arbitrarily chosen, (18) is thus guaranteed also for any symmetric queue which completes the proof of theorem 1.
Remark.

i) A similar product form expression can be given that only concerns the total queue length at each station by averaging (17) over the various job types.

ii) The results are directly applicable to open networks. To this end, one only needs to adjust the traffic equations (6) to include exterior arrivals and departures. The details are standard and therefore omitted.

5. Application Examples

In the following we present some examples satisfying (11) and (12). As we will be concerned with a fixed station $s$ we omit the subscript $s$ throughout these examples.

5.1. Blocking

i) Common Pool

Clearly, a first example of interdependent blocking of parallel queues which satisfies (11) or rather (13) and (14), is obtained by imposing a common capacity constraint or limited pool on the parallel queues, as by

$$A(q | \bar{q}) = \begin{cases} 1 & \text{if } n < M \\ 0 & \text{otherwise} \end{cases}$$

with a total number of jobs, some constant $M$ and regardless of $q$.

ii) Hierarchical Sharing

Suppose that there are 4 job types and a separate queue for each of them at a particular station $s$. No more than $M_t$ jobs of type $t$ can be stored for $t = 1, ..., 4$ but also jobs of type 1 and 2 have to share a limited device for at most $M_5$ jobs and of type 3 and 4 for at most $M_6$ while all queues jointly have to share a restricted device for no more than $M_7$ jobs.
The convexity condition (14) is directly verified with $B$, the set of all states such that
\[ n_1 + n_2 < M_6 \]
\[ n_3 + n_4 < M_7 \]
\[ n_1 + n_2 + n_3 + n_4 < M_7 \]

Note that this structure is covered by [5,16] as an entire network while assuming exponential inputs. Our description allows this structure just as a station and does not require exponentiality conditions.

iii) **Synchronous Servicing**

Again suppose that each queue is associated with a particular job-type and with a capacity constraint of $M'$ for queue $t$, for $t = 1, 2, ..., T$. Each queue however wishes to operate as an infinite server queue to which purpose it borrows the required number of servers from a central depot with a capacity of $M$ servers when possible. A type-$t$ job however requires a service by $b^t$ servers simultaneously. If upon arrival of a type-$t$ job only less than $b^t$ servers are available the job is rejected.
As above, the convexity condition (14) is verified with $B$, the set of all states such that
\[ n^t \leq M^t \quad \text{for} \quad t = 1, \ldots, T \]  
\[ n^1 b^1 + \ldots + n^T b^T \leq M \]

This system is treated by [9,16,35] as a total network model with exponential input assumptions. Here we allow such a station within a network and we do not require exponentiality for multiserver disciplines.

5.2. Servicing

i) Processor Sharing

As a first example satisfying (12) assume that each job type has a particular queue while all queues share a single processor by ratio of jobs. More precisely, with $m^q$ the number of jobs at queue $q$ and $m$ the total number of all queues for a given vector $\Pi$, (12) is guaranteed by

\[ \Phi(q | \Pi) = \frac{m^q}{m} \]  

with as corresponding product for a given vector $\Pi$ with jobs present from types $t_1, \ldots, t_c$:  

\[ n^t b^t \]
ii) **Global Acceleration**

Each queue may have an individual service rate function but excess of a threshold value by the total workload at the station, may enforce a common acceleration of all individual queues. More precisely, with \( m_s \) and \( m \) as defined above and arbitrary functions \( \Phi(m) \) and \( \Phi'(m^s) \), condition (12) is satisfied by

\[
\Phi(q | \pi) = \Phi(m) \Phi'(m^s)
\]

(42)

For a given vector \( \pi \) the product becomes

\[
[\prod_{k=1}^{m} \Phi'(k)] \prod_{k=1}^{m} \Phi'(k)
\]

(43)

For example, with

\[
\Phi(m) = \begin{cases} 
1 & \text{if } m < L \\
2 & \text{if } m \geq L
\end{cases}
\]

(44)

the service speed of each queue is doubled as soon as the total number exceeds a limit \( L \). This seems realistic in various applications.

iii) **Local Acceleration**

As another example of parallel servicing the service rate at one queue may be speeded up by each additional job at another queue. This may reflect for instance a more efficient utilization of a joint processor when more jobs are present or a more greedy use of resource when a queue views more resource competitors in its environment. For example, with 2 job types and a separate queue for each of them, one can verify the service invariance condition (12) with

\[
\Phi(1 | (n_1, n_2)) = 2^{n_2} \quad (n_1 > 0)
\]

(45)

\[
\Phi(2 | (n_1, n_2)) = 2^{n_1} \quad (n_2 > 0)
\]
8. Conclusion

A product form expression is established for queueing networks of collected parallel queues with interdependent servicing and blocking. The expression unifies various product form results such as for reversible networks with blocking, networks without blocking but station interdependent servicing and networks with mixed exponential (e.g., FCFS) and nonexponential (e.g., Processor Sharing) queues, but also allows for instance examples with non-reversible routing and blocking. A sufficient condition for this product form to hold is given in concrete terms of servicing and of blocking functions. Various practical examples such as with synchronous servicing, hierarchical blocking or resource sharing can so be concluded directly. The product form is insensitive to service distributional forms at symmetric queues. The proof is notationally complex but conceptually straightforward and self-contained as based upon different partial balance notions. Variations such as with zero service speeds and modified blocking protocols can easily be built in.
7. Table of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of Stations</td>
</tr>
<tr>
<td>$T$</td>
<td>Number of Job Types</td>
</tr>
<tr>
<td>$S$</td>
<td>Set of Symmetric Queues</td>
</tr>
<tr>
<td>$s$</td>
<td>Station number</td>
</tr>
<tr>
<td>$q$</td>
<td>Queue Number at a Station</td>
</tr>
<tr>
<td>$t$</td>
<td>Job Type Number of a Job</td>
</tr>
<tr>
<td>$r$</td>
<td>Number of Residual Exponential Service Phases of a Job</td>
</tr>
<tr>
<td>$Q(s)$</td>
<td>Number of Queues at Station $s$</td>
</tr>
<tr>
<td>$n_t$</td>
<td>Fixed Queue Number of a Type $t$ Job at Station $s$</td>
</tr>
<tr>
<td>$n_{r_t}$</td>
<td>Number of Type $t$ jobs at station $s$</td>
</tr>
<tr>
<td>$m_t$</td>
<td>Number of Jobs at Queue $q$ of station $a$</td>
</tr>
<tr>
<td>$\bar{n}_q$</td>
<td>Configuration of Certain Type Job Numbers at $a$</td>
</tr>
<tr>
<td>$\bar{m}_q$</td>
<td>Configuration of Queue Lengths at Queues of $a$</td>
</tr>
<tr>
<td>$p_{q,s}$</td>
<td>Routing Probability from Station $s$ to $s'$ and from type $t$ to $t'$</td>
</tr>
<tr>
<td>$A_s(q,</td>
<td>\bar{m}_q)$</td>
</tr>
<tr>
<td>$\Phi(q,</td>
<td>\bar{m}_q)$</td>
</tr>
<tr>
<td>$\delta_s(p,</td>
<td>\bar{m}_q)$</td>
</tr>
<tr>
<td>$\Gamma_s(p,</td>
<td>\bar{m}_q)$</td>
</tr>
<tr>
<td>$\nu_q$</td>
<td>Exponential Phase Parameter of Type $t$ job at station $s \in S$</td>
</tr>
<tr>
<td>$\tau_q$</td>
<td>Mean Service Requirement of Type $t$ job at station $s \in S$</td>
</tr>
<tr>
<td>$\mu_{r,t}$</td>
<td>Exponential Service Parameter at Queue $q$ of station $s \in S$</td>
</tr>
<tr>
<td>$\sigma_s(k)$</td>
<td>Probability of a Type $t$ job at station $s \in S$ for $k$ phases</td>
</tr>
<tr>
<td>$\lambda_q$</td>
<td>Traffic Parameter of a Type $t$ job at station $s$</td>
</tr>
</tbody>
</table>
8. References


