Traffic Relationships in Networks of Tasks

C.M. Woodside, G.M. Yee
Telecommunications Research Institute of Ontario
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada, K1S 5B6

February 18, 1998

Abstract

We consider a network of communicating software tasks in a distributed system. Over the period of interest they are permanent concurrent entities, statically allocated to processors. Tasks communicate by messages sent to ports, sockets, or entries. A task execution is such that for each message received on a certain port, a known average number of other actions take place, including sending of messages and waiting for replies, or reception of messages on some designated port. Various interesting special cases occur. From the behaviour model and parameters of each task, network-wide relationships are obtained that govern the rates of intertask messaging. Certain rates may be taken as independent and they determine the remaining rates. In some cases, a narrowly constrained operating subspace results which contains all feasible task throughput combinations. The relationships cover a variety of types of tasks, and types of communication mechanisms, including both rendezvous and non-blocking send with blocking receive.

1 Introduction

Objectives

A distributed software program may run on many computers simultaneously, with separate tasks forming part of the same application and communicating across a network. The application may include parallel computations, data bases, real-time sensor interfaces, and operator displays. A survey of the literature on software for such multi-task applications is given by Andrews and Schneider in [1].

Messaging is often the communication paradigm adopted for such systems. Messaging gives rise to traffic which, as we shall see, is determined by simple quantitative cause-effect flow relationships. Various forms of flow relationships may arise due to different styles of messaging (reflecting architectural forms). The objectives of this paper are to present some of these relationships for both synchronous (rendezvous) and asynchronous messaging. The relationships express the throughputs of a group of tasks in terms of the throughputs of certain independent "reference" tasks and determine the feasibility of contemplated message flows on a network-wide basis. Infeasible flows are found, for instance, when there is an explosion of messages around a messaging loop, and may imply the presence of a temporal error.

The relationships may be applied, for example, to the two alternate task messaging designs shown in Figure 1.1. We denote tasks by parallelograms and message paths by arrows from the sender task to the receiver task. An example of the questions we wish to answer is, "given required mean throughputs for tasks 6 and 7 (as well as the messaging requirements), which design will satisfy the requirements with feasible flows?"

Software tasks as originators of message traffic have not been much studied, but there are several sources of related work. Most data communications traffic models assume simple exogenous (often Poisson) traffic sources without internal behavioural detail (e.g. Kleinrock [2]). Models of window flow control (e.g. Reiser [3]) go further and usually include a model of a user task on either end of a connection, but these models are normally restricted to two-party connections only, and often only have unidirectional producer-consumer relationships between the user tasks.

The traffic relationships presented below are an extension of similar relationships in classical queuing networks (e.g. Jackson [4], Gordon and Newell [5]) and computer and communications performance models (e.g. Sauer and Chandy [6]). However, task messages do not represent customers in a queuing network because messages are not conserved; a message arriving at a task may stimulate zero, one, or several further messages. In this aspect, messages are more similar to procedure call frequencies in software models (e.g. [7]).

Other models of distributed software (e.g. Heidelberger and Trivedi [8], Thomasian and Bay [9]) have created artificial populations of customers with special behaviour which simulate the variability in the message populations.

Some previous work by Woodside et al [10, 11] has examined special cases of task networks involving rendezvous only. It was found that task traffic relationships form a coordinating set of flow constraints which bind together the separate components in a performance model. This paper extends the previous work by concentrating on the traffic analysis, and introducing two important new features which occur frequently in practical systems – selection of a port or set of ports by the receiver task, and asynchronous message sending.

Chinneck [12] has formulated an abstract version of this problem which he calls "processing networks" in which he obtained flow relationships similar to those in this paper, and extensive results about network behaviour. However, the communication model here (particularly the RA/RS port behaviour) is not exactly the same as in the model in [12]. This paper may be regarded as a special case of [12] with some extensions that arise from the application.

Tasks

Tasks, as the term is used here, are concurrent units of execution (also call processes in other works). They are statically allocated to processors and cycle forever. Tasks request and perform services (client/server model) by sending and receiving messages among themselves [1]. Message **traffic** is the rate of flow of such messages between tasks. In servicing a request, a task may itself require service from other tasks, before completing its own service. Messages are sent and received at communication channel interface points called **ports** (denoted by small parallelograms inlaid into the task parallelogram boundary).

In the client/server model, tasks may be clients (send but not receive), servers (receive but not send), or client-servers (both send and receive). Server and client-server tasks have mean message serving cycle rates or throughputs associated with every receive port and with the task as a whole. Client-server tasks send a certain mean number of messages per message received. Client tasks have a mean cycle rate or throughput associated with the task as a whole and make a mean number of calls per cycle. For clarity we will use the term "throughput" rather than both "throughput" and "cycle rate".

Task Behaviour Model

We consider both synchronous and asynchronous communication mechanisms. The synchronous mechanism (rendezvous) is implemented by a blocking send or **send-wait-reply** sequence, in conjunction with a blocking receive, **receive-wait**. The asynchronous mechanism is implemented by a non-blocking send, **send-no-wait** (with non-blocking **reply** by the receiver) in conjunction with blocking receive **receive-wait** [14, 1, 10].

Task **port receive behaviour policies** describe the order and probability of selecting various combinations of ports on which to receive. We will study the following receive policies: **receive-specific** (task selects to receive at a specific port with a certain probability), **receive-any** (task selects to receive at any port within a set of ports with a certain probability), **receive-any/receive-specific** (task selects to receive at any port within a set of ports with a certain probability or alternatively it selects to receive at specific ports within the set with certain probabilities). The selection probabilities are steady state probabilities. Start-up port selection probabilities may be independent of previously selected ports or dependent on the previous selection (Markovian).

We deal with task networks which may be **open** or **closed**, **cyclic** or **acyclic**, and which may be **feasible** or **infeasible** with respect to traffic flows. A description of various task network architectures is given in [13].

We shall term a network **feasible** if 1) task message throughputs do not violate any traffic relationships imposed by the task behaviour model, and 2) task message throughputs must not all be zero. An **infeasible** network is one which is not feasible, as in Figure 1.2. In the Figure, the throughput of each receiving task is driven by the throughputs of its senders. Task 2 receives 1 message per cycle of task 1, and therefore has a throughput equal to that of task 1. However, task 1 receives 2 messages per cycle of task 2, and therefore has a throughput twice that of task 2 – a contradiction! (In [12] the term viable was used instead of feasible.)

Section 2 gives further details on the task behaviour model and presents corresponding traffic relationships. Section 3 considers the solution of these relationships. Section 4 presents conclusions.

2 Traffic Relationships

Notation

Define:

i - task i;

$i.m$ - port $i.m$ of task i, $m \geq 0$; task i is a client task if and only if $m = 0$, and $i.0$ denotes a virtual port from which task i calls other tasks;

$i.\mathcal{S}$ - set $i.\mathcal{S}$ of ports of task i; different sets will be assumed to be disjoint, i.e. $i.\mathcal{S} \cap i.\mathcal{T} = \phi$ for any two sets $i.\mathcal{S}$ and $i.\mathcal{T}$;

$Y_{src,dst}$ - mean number of messages from src (source) to dst (destination) per message received at src (per task cycle if src is a client task), where src and dst are of the form i, $i.m$, or $i.\mathcal{S}$ above;

λ_{loc} - mean message throughput of loc (location) where loc is of the form i, $i.m$, or $i.\mathcal{S}$ above; if i is a client task, then $\lambda_i \equiv \lambda_{i.0}$ is the same as the task's mean cycle rate;

$\alpha_{i.m}$, $\alpha_{i.\mathcal{S}}$ - steady state probability of task i selecting to receive at $i.m$, $i.\mathcal{S}$ respectively (defined only for receive-specific and receive-any ports, respectively, such that

$$\sum_{i.m} \alpha_{i.m} + \sum_{i.\mathcal{S}} \alpha_{i.\mathcal{S}} = 1 \ .$$

Task Behaviour Model

We express the task behaviour model in terms of communication primitives and port receive behaviour models.

Communication Primitives

The primitives implementing the synchronous and asynchronous communication mechanisms are:

- **send-wait-reply (SWR)** - (blocking) sender waits at the receiver's receive port for a reply from the receiver; the reply is executed by the receiver and may consist of a simple confirmation of message received or may be the actual results of the request;

- **receive-wait (RW)** - (blocking) receiver waits at its receive port for a message from any sender;

- **send-no-wait (SNW)** - (non-blocking) sender sends a message to the receiver and returns immediately; the message is not lost if the receiver is not ready to receive it as it is copied to the receiver's message-in buffer (to be taken by the receiver later); the results of the message are replied back to the sender at a later time using the **reply** primitive;

- **reply** - (non-blocking) receiver returns to the sender the results or a receive confirmation of a send-no-wait message.

We do not model the receiver replies in either the rendezvous or send-no-wait/receive-wait cases; we assume that replies cause negligible delay.

Port Receive Behaviour Policies

Three port receive behaviour policies will be considered. Within each set of ports, one of the following policies is used:

- **Receive-Specific (RS)** - (Figure 2.1a (rectangles with X denote task mainlines)) a task i selects to receive at a specific $i.k$ with probability $\alpha_{i.k}$ (equivalent to the Ada Accept);

- **Receive-Any (RA)** - (Figure 2.1b) a task i selects to receive at any port (first port with a message) contained within $i.\mathcal{S}$ with probability $\alpha_{i.\mathcal{S}}$ (equivalent to the Ada Selective Accept with no guards);

- **Receive-Any/Receive-Specific (RA/RS)** - (Figure 2.1c) a task i selects to receive at any port within $i.\mathcal{S}$ with probability $\alpha_{i.\mathcal{S}}$ **or** selects to receive at a specific port $i.k$ within $i.\mathcal{S}$ with probability $\alpha_{i.k}$. We label port $i.k$ a RA/RS port.

RS is a straightforward execution of a receive on one port, followed by processing associated with that port. RA is like an Ada "select" statement, in which the actual case followed depends on what port first receives a message. RA/RS combines the two.

The port selection may be independent of the previously selected ports or dependent only on the previous selection. In the latter case, the set of task states corresponding to port choices results in a Markov chain. We assume that the steady state probabilities of this Markov chain exist and can be found by standard methods; they are exactly the probabilities α described above.

Relationships for SWR-RW (Rendezvous) Communication

The basic quantities of interest are the individual port throughputs $\lambda_{i.k}$ in messages/sec. From these, other measures such as task or subnet throughputs can be derived.

The above task behaviour model gives the following traffic relationships for synchronous rendezvous communication:

For all tasks with RS ports $i.m$, $j.n$ and/or RA port sets $i.\mathcal{S}$, $j.\mathcal{T}$, the port throughputs satisfy

1. Non-negativity

$$\lambda_{i.m} \geq 0 \; ; \qquad Y_{i.m,j.n} \geq 0 \; , \qquad n \neq 0 \; . \qquad (1)$$

2. Call Additivity

$$Y_{i.m,j} = \sum_{n \neq 0} Y_{i.m,j.n} \; , \qquad (2)$$

$$Y_{i.m,j.\mathcal{T}} = \sum_{n \in j.\mathcal{T}} Y_{i.m,j.n} \; , \qquad (3)$$

$$\lambda_i Y_{i,j} = \sum_m \sum_{n \neq 0} \lambda_{i.m} Y_{i.m,j.n} \; , \qquad (4)$$

$$\lambda_{i.\mathcal{S}} Y_{i.\mathcal{S},j.n} = \sum_{m \in i.\mathcal{S}} \lambda_{i.m} Y_{i.m,j.n} \; , \qquad n \neq 0 \; . \qquad (5)$$

3. Throughput Additivity

$$\lambda_j = \sum_{n \neq 0} \lambda_{j.n} \; , \qquad (6)$$

$$\lambda_{j.\mathcal{T}} = \sum_{n \in j.\mathcal{T}} \lambda_{j.n} \; . \qquad (7)$$

4. Throughput Composition

$$\lambda_{j.n} = \sum_{i \neq j} \sum_{m} \lambda_{i.m} Y_{i.m,j.n} \ , \qquad n \neq 0 \ . \tag{8}$$

5. Throughput Proportionality

$$\begin{aligned} \lambda_{j.n} &= \alpha_{j.n} \lambda_j \ , \qquad n \neq 0 \ , & (9) \\ \lambda_{j.\mathcal{T}} &= \alpha_{j.\mathcal{T}} \lambda_j \ . & (10) \end{aligned}$$

For all tasks with RA/RS port set $j.\mathcal{T}$, (1)–(8) hold. However, (9) and (10) must now be combined for RA/RS port $j.n \in j.\mathcal{T}$. For such a $j.n$ we now have

$$\lambda_{j.n} = (\lambda \text{ due to RS}) + (\lambda \text{ due to RA}) \ .$$

This gives

$$\alpha_{j.n} \leq \frac{\lambda_{j.n}}{\lambda_j} \leq \alpha_{j.n} + \alpha_{j.\mathcal{T}} \ , \qquad n \neq 0 \tag{11}$$

using (9) and (10).

Geometric Interpretation

If we plot one port throughput against another for the same task, we can see the consequences of (1)–(10) more clearly. For any task with RS ports $j.k$, $j.n$, (9) gives

$$\lambda_{j.n} = \frac{\alpha_{j.n}}{\alpha_{j.k}} \lambda_{j.k} \ , \qquad n \neq 0 \ . \tag{12}$$

This confines $(\lambda_{j.k}, \lambda_{j.n})$ to a straight line (Figure 2.2a). For a task with RA port sets $j.\mathcal{S}$, $j.\mathcal{T}$, (10) gives

$$\lambda_{j.\mathcal{S}} = \frac{\alpha_{j.\mathcal{S}}}{\alpha_{j.\mathcal{T}}} \lambda_{j.\mathcal{T}} \ . \tag{13}$$

This confines $(\lambda_{j.\mathcal{T}}, \lambda_{j.\mathcal{S}})$ to a straight line (Figure 2.2b).

Let $j.\mathcal{T}$ be any RA/RS set belonging to task j. Then for every RA/RS port $j.n \in j.\mathcal{T}$, every RS port $j.k \notin j.\mathcal{T}$, and every RA set $j.\mathcal{S}$, inequality (11) gives (note $n \neq 0$ in the next 3 expressions)

$$\frac{\alpha_{j.n}}{\alpha_{j.k}} \leq \frac{\lambda_{j.n}}{\lambda_{j.k}} \leq \frac{\alpha_{j.n} + \alpha_{j.\mathcal{T}}}{\alpha_{j.k}} \ , \qquad \alpha_{j.k} > 0 \ , \tag{14}$$

and

$$\frac{\alpha_{j.n}}{\alpha_{j.\mathcal{S}}} \le \frac{\lambda_{j.n}}{\lambda_{j.\mathcal{S}}} \le \frac{\alpha_{j.n} + \alpha_{j.\mathcal{T}}}{\alpha_{j.\mathcal{S}}} , \qquad \alpha_{j.\mathcal{S}} > 0 . \qquad (15)$$

For RA/RS ports $j.n$ and $j.k$ contained within $j.\mathcal{T}$, inequality (11) gives

$$\frac{\alpha_{j.n}}{\alpha_{j.k} + \alpha_{j.\mathcal{T}}} \le \frac{\lambda_{j.n}}{\lambda_{j.k}} \le \frac{\alpha_{j.n} + \alpha_{j.\mathcal{T}}}{\alpha_{j.k}} , \qquad \alpha_{j.k} > 0 . \qquad (16)$$

Inequality (16) confines $(\lambda_{j.k}, \lambda_{j.n})$ to the region R shown in Figure 2.2c. In fact, inequalities (11), and (14) – (16) confine the vector $(\lambda_{j.k}, \ldots, \lambda_{j.\mathcal{S}}, \ldots, \lambda_j)$ to a hypercone with the "tip" at the origin in $K+1$ space, where K is the number of port throughputs of task j (plus 1 for λ_j). The hypercone extends away from the origin satisfying the non-negativity constraints for λ.

The effect of inequalities (11) and (14) – (16) is network-wide due to relation (8). The throughputs of all tasks are confined to subspaces. This is easily seen from (8) and (11):

$$\alpha_{j.n} \le \frac{1}{\lambda_j} \sum_{i \ne j} \sum_{m} \lambda_{i.m} Y_{i.m,j.n} \le \alpha_{j.n} + \alpha_{j.\mathcal{T}} , \qquad n \ne 0 .$$

Relationships for SNW-RW Communication

For the task behaviour model above, the traffic relationships for rendezvous also hold for asynchronous SNW-RW. This is arrived at as follows:

Consider a task using SNW-RW communication first as a sender. The throughput of such a sender still depends on the message service times of its receivers. Eventhough this sender returns immediately to its execution after sending, it cannot get too far ahead without reaching some sort of limit, e.g. it cannot proceed without certain results or message buffer overflow (we disallow the sender over-running the receiver with messages, as a realistic requirement).

On average, the throughput of such a sender may be thought of as being equivalent to the throughput of a sender with SWR sends of "appropriate" wait times (due to the restrictions against the sender getting too far ahead). That is, the sender continuously sends messages until it can go no further; it therefore waits for a time T after which it can continue its execution. The time T may be thought of as being partitioned into appropriate wait times and assigned to each of the earlier sends when the sender did not wait. It follows that for two senders, identical except for the fact that one uses SNW and the other uses SWR, we have

$$\lambda_{SWR} \leq \lambda_{SNW} \tag{17}$$

since it is clear that $T \leq T_0$ where T_0 is the sum of all the wait times of the corresponding sends for the SWR sender.

As a sender task, considering that the SNW's may be thought of as being equivalent to SWR's of appropriate wait times, this sender behaves like the rendezvous case. As a receiver task, the communication mechanism **is** RW, the same as in the rendezvous case. Therefore, the traffic relationships of this case are the same as the ones for the rendezvous case. However, the message throughputs here will be equal to or greater than the corresponding rendezvous throughputs by inequality (17).

Application to Open and Closed Networks

The above relationships apply equally to open and closed task networks. For open networks, outside arrivals may be included in (8) provided only that they are quantifiable in the form $\lambda_{outside} Y_{outside,j.n}$.

Application to Tasks with Different Port Models

Interesting task networks are those in which a task may receive on all the port receive policies described in the task behaviour model. Which traffic relationships apply to this task? The key relationships are (1), (8), (9), (10) and (11). Clearly (1) and (8) apply to all ports of this task, whereas (9), (10) and (11) apply to RS, RA, and RA/RS ports respectively. Applying the key relationships to all such tasks in the network (and outside an open network) results in systems of linear equations and inequalities on throughputs, which may be solved to determine the feasibility of traffic flows (section 3).

All other expressions are defining or derived relationships. The derived ones (e.g. relationships (12) – (16)) describe task behaviour.

Application Example - Resource Sharing

As a practical application of the above relationships, consider a task network in which client tasks send messages to server tasks to request access to shared resources. The server tasks control access to the resources by means of accepting or not accepting calls for resource tokens (stored internally) based on token availability. We may apply the above task behaviour model to this situation and solve the traffic relationships to determine the feasibility of envisaged resource requests.

3 Solution for Traffic Relationships

This section summarizes some computations which may be used to explore the properties of a particular system. More extensive results applicable to many of these systems are given in [12].

We begin with a simple example of calculating the message throughputs of a proposed design (Figure 3.1) using the key traffic relationships of section 2. In Figure 3.1, set $\mathcal{S}$ is RA, set $\mathcal{T}$ is RA/RS, and port 3.1 is RS. The following data is given: $Y_{1.0,3.1} = Y_{1.0,3.2} = Y_{1.0,2.1} = Y_{1.0,2.2} = 1$, $Y_{2.1,3.3} = 2$, $\alpha_{3.\mathcal{T}} = 0.4$, $\alpha_{3.1} = 0.3$, $\alpha_{3.2} = 0.2$, $\alpha_{3.3} = 0.1$, $\alpha_{2.\mathcal{S}} = 1$. Thus (1) further requires $\lambda_{1.0}$, $\lambda_{2.1}$, $\lambda_{2.2}$, $\lambda_{3.1}$, $\lambda_{3.2}$, and $\lambda_{3.3}$ to be all non-negative. Relationship (8) gives

$$\begin{aligned} \lambda_{3.1} &= \lambda_{1.0} \,, \qquad \lambda_{3.3} = 2\lambda_{2.1} \,, \qquad \lambda_{2.2} = \lambda_{1.0} \,, \\ \lambda_{3.2} &= \lambda_{1.0} \,, \qquad \lambda_{2.1} = \lambda_{1.0} \,. \end{aligned}$$

Relationship (9) yields

$$\lambda_{3.1} = 0.3(\lambda_{3.1} + \lambda_{3.2} + \lambda_{3.3}) \,.$$

Relationship (10) gives

$$\lambda_{2.\mathcal{S}} = 1 \cdot (\lambda_{2.1} + \lambda_{2.2}) \,.$$

Relationship (11) yields

$$\begin{aligned} \lambda_{3.1} &\geq 0.3(\lambda_{3.1} + \lambda_{3.2} + \lambda_{3.3}) \,, \\ \lambda_{3.1} &\leq (0.3 + 0.4)(\lambda_{3.1} + \lambda_{3.2} + \lambda_{3.3}) \,, \\ \lambda_{3.2} &\geq 0.2(\lambda_{3.1} + \lambda_{3.2} + \lambda_{3.3}) \,, \\ \lambda_{3.2} &\leq (0.2 + 0.4)(\lambda_{3.1} + \lambda_{3.2} + \lambda_{3.3}) \,. \end{aligned}$$

Figure 3.1 is simple enough that the solution may be obtained by inspection. However, the application of the key relationships to this system finds it to be infeasible, since (9) is in conflict with (8). Relationship (8) gives $\lambda_{3.1} = \lambda_{1.0}$ but relationships (8) and (9) give $\lambda_{3.1} = (1.2)\lambda_{1.0}$. The system becomes feasible by changing $\alpha_{3.1}$, $\alpha_{3.2}$ to 0.25 each. The solution is then $\lambda_{3.1} = \lambda_{3.2} = \lambda_{2.1} = \lambda_{2.2} = \lambda_{1.0}$, $\lambda_{2.\mathcal{S}} = \lambda_{3.3} = 2\lambda_{1.0}$ with $\lambda_{1.0}$ as a reference throughput whose value may be set.

This example shows how the key relationships of section 2 may be used to determine the feasibility (may not be obvious) of a task network design. In this case, eventhough the α's of task 3 summed to 1 locally, their values were also subject to the network-wide constraining effect of the traffic relationships.

We next give a formal method for calculating the port message throughput of each task in a network.

Homogeneous Systems

We first re-arrange the key relationships (8) – (11) as a homogeneous system of linear equations with non-negativity constraints for λ (from (1)) and slack variables $s_{1,j.n}$ and $s_{2,j.n}$ (for (11)). The nonnegativity of the Y's is always satisfied as given data.

$$-\lambda_{j.n} + \sum_{i \neq j} \sum_{m} \lambda_{i.m} Y_{i.m,j.n} = 0, \qquad n \neq 0 \ , \tag{18}$$

$$-\lambda_{j.n} + \alpha_{j.n} \sum_{k \neq 0} \lambda_{j.k} = 0, \qquad n \neq 0 \ , \tag{19}$$

$$-\lambda_{j.\mathcal{T}} + \alpha_{j.\mathcal{T}} \sum_{k \neq 0} \lambda_{j.k} = 0 \ , \tag{20}$$

$$\lambda_{j.n} - \alpha_{j.n} \sum_{k \neq 0} \lambda_{j.k} - s_{1,j.n} = 0 \ , \qquad n \neq 0 \ , \tag{21}$$

$$-\lambda_{j.n} + (\alpha_{j.n} + \alpha_{j.\mathcal{T}}) \sum_{k \neq 0} \lambda_{j.k} - s_{2,j.n} = 0 \ , \qquad n \neq 0 \ , \tag{22}$$

$$\lambda_{j.n} \geq 0 \ , \tag{23}$$

$$s_{1,j.n} \geq 0 \ , \qquad n \neq 0 \ , \tag{24}$$

$$s_{2,j.n} \geq 0 \ , \qquad n \neq 0 \ . \tag{25}$$

Clearly (18)–(22) make up a homogenous system of linear equations which we may write in matrix notation as

$$\mathbf{A} \begin{pmatrix} \lambda \\ \mathbf{s} \end{pmatrix} = \mathbf{0} \tag{26}$$

where $\mathbf{A}$ is the coefficient matrix, λ is the vector of port message throughputs, and $\mathbf{s}$ is the vector of slack variables (24) and (25). Let M be the total number of port equations from (18) –(22). Let $N = N_\lambda + N_s$ where N_λ is the total number of port message throughputs and N_s is the total number of slack variables. Then $\mathbf{A}$ is of size $M \times N$, λ is of size $N_\lambda \times 1$ and $\mathbf{s}$ is of size $N_s \times 1$.

For the following solution method, we assume $M = N$, since if $M > N$, (26) has no solution, and if $M < N$, rows of zeroes may be added to $\mathbf{A}$ until $M = N$. Also, let $\mathbf{g} = (\mathbf{f}, \mathbf{s})^\top = (\mathbf{g_1}, \mathbf{g_2})^\top$ and ψ a mapping from the indices of λ and $\mathbf{s}$ to the indices of $\mathbf{g}$.

Equation (26) may be solved by transforming the coefficient matrix to reduced row-echelon form (RRF). The solution space of (26) has dimension $N - k$ where k is the rank of the coefficient matrix $\mathbf{A}$. If $k = N$, the only solution is $\mathbf{g} = \mathbf{0}$. If $k < N$, a basis for the solution space can be constructed from the RRF of $\mathbf{A}$ as follows [15]:

Equation (26) is equivalent to

$$\begin{pmatrix} \mathbf{I_k} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{g_1} \\ \mathbf{g_2} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}$$

where the leftmost matrix is the RRF of $\mathbf{A}$ and $\mathbf{B}$ is of size $k \times (N-k)$, $\mathbf{g_1}$ is $k \times 1$, $\mathbf{g_2}$ is $(N-k) \times 1$. This gives

$$\mathbf{g_1} = -\mathbf{B}\mathbf{g_2} \tag{27}$$

where $\mathbf{g_2}$ may be chosen arbitrarily from nonnegative values (constraints (23) – (25)). Suppose $\mathbf{g_2} = (\beta_1, \ldots, \beta_{N-k})^\top$. Let $\mathbf{e_j}$ be the $(N-k) \times 1$ unit vector with the jth element equal to 1 and all the other elements 0. Let $\mathbf{b_j}$ be the jth column vector of $\mathbf{B}$ and $\mathbf{u_j} = (-\mathbf{b_j}, \mathbf{e_j})^\top$. Every solution of (26) can then be written in the form

$$\mathbf{g} = \beta_1 \mathbf{u_1} + \beta_2 \mathbf{u_2} + \cdots + \beta_{N-k} \mathbf{u_{N-k}} \tag{28}$$

The $\mathbf{u_j}$ form a basis for the solution space of (26). Notice that if $k = N$, then $\mathbf{B}$ is nonexistent and the only solution is $\mathbf{g} = \mathbf{0}$.

Having obtained a basis we generate solutions to (26) and check them for non-negativity (required by (23) – (25)) to obtain a nonzero feasible solution.

Reference Message Throughputs

Unknown port throughputs may be expressed in terms of independent **reference throughputs** (values which may be set) by using (27), which expresses the unknown throughputs of $\mathbf{g_1}$ in terms of the reference throughputs of $\mathbf{g_2}$. A **reference task** is defined to be a task with only reference throughputs.

The number of reference throughputs is bounded above by $N-k$ which is the number of independent basis vectors (degrees of freedom) in the solution space. Physically, this means that certain throughputs are determined by the throughputs of at most $N-k$ independent reference tasks, corresponding to the reference throughputs – a consequence of calling pattern. While client tasks are definite candidates for the role of reference tasks, they do not preclude other types of tasks from being reference tasks in a complex network.

We extend the solution method in section 3.1 to allow identification of reference throughputs as follows: Define $i.*$ as either $i.m$ or $i.\mathcal{S}$. Choose j such that g_{k+j} is not a slack variable (recall k is the rank of $\mathbf{A}$). Then reference throughput $\lambda_{i.*} = g_{k+j} = \beta_j$ where $i.* = \psi^{-1}(k+j)$ gives the required identification.

It may arise that a desired reference throughput has index $i.*$ with $1 \leq \psi(i.*) \leq k$ (and hence not among the independent values in the above solution method). It is sufficient here to relabel the throughputs so that the desired throughput has index $i.*$ with $k < \psi(i.*) \leq N$ and re-solve.

Chinneck's Test for Feasibility

The feasibility of λ requires that $\lambda \geq \mathbf{0}$, $\mathbf{s} \geq \mathbf{0}$. Because of the homogeneity of (26), the feasible throughputs are a cone with its vertex at $\mathbf{0}$; if $(\lambda, \mathbf{s})^\top$ satisfies (26) then so does $(c\lambda, c\mathbf{s})^\top$ for constant c. A simple practical test for feasibility, which is given in [12], is to add to (18) – (25) the constraint $\lambda \cdot \mathbf{1} = constant > 0$ (the sum of all throughputs is constrained to any positive constant; this is added to exclude $\mathbf{0}$ which always satisfies (26)) and to apply a phase-1 simplex algorithm to determine a feasible point. The phase-1 search is described in any book on linear programming (e.g. [16]). If the phase-1 search fails, then there is a problem akin to the loop explosion in Figure 1.2.

Graph Theoretic Interpretations

We are interested in 1) how the rank of $\mathbf{A}$ relates to its corresponding task network and 2) network architectures that result in only all-zero solutions for (26).

The row rank of a matrix is its number of linearly independent rows. Linearly independent rows correspond to linearly independent throughput expressions. That is, the network calling pattern is such that these expressions are unique and not just linear combinations of one another.

If (26) gives only the all-zero solution or negative throughputs/slack variables, then the corresponding task network has infeasible traffic flows. Clearly the values of traffic parameters (e.g. Y's) in the network determine this but certain network calling patterns may also lead to such solutions as well. For example, the distinct networks in Figure 3.2 have only all-zero solutions.

4 Conclusions

Given the task behaviour model, message flows in a task network employing synchronous rendezvous communication or asynchronous SNW-RW communication, or both, are constrained by the traffic relationships of section 2. Together with the solution method in section 3 these relationships offer a way to practically validate the correctness and performance of distributed task designs (e.g. resource sharing). The results are closely related to recent work in Operations Research on "processing networks".

Using the traffic analysis of this paper, a compact representation of the throughputs for a large application can be found in terms only of the independent reference throughputs, and the number of these independent throughputs can be determined. Using a model for execution, communication, and contention delays, a complete performance model can be built up with the traffic analysis as a foundation. The traffic equations give a set of coordinating relationships for the performance model. In a limited degree this was used personally in [10, 11].

Acknowledgements

Conversations with John Neilson and Dorina Petriu were helpful in formulating this paper, and John Chinneck told us about processing networks and the properties he had discovered. The financial support of Bell-Northern Research and the National Sciences and Engineering Research Council of Canada for Mr. Yee's scholarship, and of NSERCC and the Telecommunications Research Institute of Ontario for other aspects of the research, is gratefully acknowledged.

References

[1] Andrews, G.R.; Schneider, F.B.; "Concepts and Notations for Concurrent Programming"; ACM Computing Surveys, Vol. 15, No. 1, March 1983.

[2] Kleinrock, L.; "Queuing Systems Volume II: Computer Applications"; John Wiley and Sons, 1976.

[3] Reiser, M.; "A Queueing Analysis of Computer Communication Networks with Window Flow Control", I.E.E.E. Trans. on Communications, Vol. COM-27, pp. 1199–1209, August 1979.

[4] Jackson, J.R.; "Jobshop-Like Queueing Systems", Management Science 10, 1 (October 1963) pp. 131–142.

[5] Gordon, W.J.; Newell, G.F.; "Closed Queueing Networks with Exponential Servers", Operations Research 15, pp. 244-265, 1967.

[6] Sauer, C.H.; Chandy, K.M.; "Computer Systems Performance Modeling", Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1981.

[7] Smith, C.U.; "The Prediction and Evaluation of the Performance of Software from Extended Design Specification", University of Texas, August 1980.

[8] Heidelberger, P.; Trivedi, K.S.; "Analytic Queueing Models for Programs with Internal Concurrency", I.E.E.E. Transactions on Computers, Vol. C-32, No. 1, January 1983.

[9] Thomasian, A.; Bay, P.; "Analytic Queuing Network Models for Parallel Processing of Task Systems", I.E.E.E. Transactions on Computers, Vol. C-35, No. 12, December 1986.

[10] Woodside, C.M.;"Throughput Calculations for Basic Stochastic Rendezvous Networks"; Report 86-10, Department of Systems and Computer Engineering, Carleton University, Ottawa, May 1986, in press in *Performance Evaluation.*

[11] Miernik, J.W.; Woodside, C.M.; Neilson, J.E.; Petrui, D.C.; "Throughput of Stochastic Rendezvous Networks with Caller-Specific Source and Processor Contention"; Proc. IEEE Infocom '88, pp. 1040-1049, March 1988.

[12] Chinneck, J.W.; "Formulating Processing Networks: Viability Theory"; submitted to Naval Research Logistics, 1988.

[13] Buhr, R.J.A.; "System Design with Ada"; Prentice Hall, 1984.

[14] Gentleman, W.M.; "Using the Harmony Operating System"; ERB-966, NRCC No. 27469; National Research Council of Canada, Ottawa, 1987.

[15] Noble, B.; "Applied Linear Algebra"; Prentice Hall, 1969.

[16] Simmons, D.M.; "Linear Programming for Operations Research"; Holden-Day, 1972.