On the Performance of Synchronized Programs in Distributed Networks with Random Processing Times and Transmission Delays

Sergio Rajsbaum* and Moshe Sidi †
Technion – Israel Institute of Technology
Haifa, Israel 32000

September 15, 1992

Abstract

A synchronizer is a compiler that transforms a program designed to run in a synchronous network into a program that runs in an asynchronous network. The behavior of a simple synchronizer, which also represents a basic mechanism for distributed computing and for the analysis of marked graphs, was studied in [ER1] and [ER2] under the assumption that message transmission delays and processing times are constant. In this paper we study the behavior of the simple synchronizer when processing times and transmission delays are random. Our main performance measure is the *rate* of a network, i.e., the average number of computational steps executed by a processor in the network, per unit time. We analyze the effect of the topology and the probability distributions of the random variables on the behavior of the network. For random variables with exponential distribution we provide tight (i.e. attainable) bounds and study the effect of a bottleneck processor on the rate.

1 INTRODUCTION

Consider a network of processors which communicate by sending messages along communication links. The network is *synchronous* if there is a global clock whose beats are heard by all the processors simultaneously, and the time interval between clock beats is long enough for all messages to reach their destinations and for local *computational steps* to be completed before the clock beats again. The network is *asynchronous* if there is no global clock, and the transmission times of messages are unpredictable.

*Computer Science Department. Present address: Instituto de Matematicas–UNAM, Ciudad Universitaria, D.F. 04510, Mexico. (rajsbaum@redvax1.dgsca.unam.mx)
†Electrical Engineering Department (moshe@techsel)

In general, a program designed for a synchronous network will not run correctly in an asynchronous network. Instead of designing a new program for the asynchronous network, it is possible to use a *synchronizer,* [A1], i.e., a compiler that converts a program designed for a synchronous network, to run correctly in an asynchronous network. Synchronizers provide a useful tool because programs for synchronous networks are easier to design, debug and test than programs for asynchronous networks. Furthermore, an important use of synchronizers is the design of more efficient asynchronous algorithms [A2]. The problem of designing efficient synchronizers has been studied in the past (e.g. [A1], [AP90], [PU89]).

The (worst case) time complexity of a distributed algorithm is usually computed assuming that processing times and message transmission delays are equal to some constant which represents an upper bound on these durations. The goal of this paper is to study the effect of *random* processing times and transmission delays on the performance of synchronous programs running in an asynchronous network under the control of a simple synchronizer. We compare the results with the deterministic case [ER1], [ER2], in which processing times, as well as message delays, are constant (or bounded).

The operation of the synchronizer is as follows: Each processor waits for a message to arrive on each of its in-coming links before performing the next computational step. When a computational step is completed (after a random time), it sends one message on each of its out-going links. The implementation of this synchronizer may require, for instance, that every message is followed by an end-of-message marker, even if the message is empty. These end-of-message markers model the flow of information that must exist between every pair of processors connected by a link in each computational step [A2]. This is how a processor knows it has to wait for a message which was sent to it, or if no message was sent.

We use this synchronizer in our analysis since it is very simple, yet, it captures the essence of the synchronizer methodology, i.e., it ensures that a processor does not initiate a new phase of computation before knowing that all the messages sent to it during the previous phase have already arrived. Moreover, the synchronizer is equivalent to a marked graph (e.g. [CHEP]) in which the initial marking has one token per edge. In [Ra91] and [RM92] the relationship between synchronizers and marked graphs is studied, and it is shown how the simple synchronizer can model the behavior of any marked graph, of the synchronizers of [A1], and of distributed schedulers in [BG89], [MMZ88]. Thus, our work is closely related to problems in stochastic petri nets, where, due to the huge size of the state space, the solution techniques often rely on simulation (e.g. [M1], [M2], [Ma89]).

Many distributed protocols are based on this simple synchronizer. For example, the snapshot algorithm [CL85], clock synchronization algorithms (e.g. [BS88], [OG87]), the synchronizers of [A1], the distributed schedulers in [BG89], [MMZ88], the *optimistic* synchronizer [GRST92]. The synchronizer is similar to synchronizer α in [A1], but can be be used also in directed networks, as opposed to other synchronizers suggested in [A1] that require all links to be bidirectional. In [ER1] and [ER2] the benefits of using the synchronizer as an initialization procedure are described.

Main Results

This paper is devoted to the performance analysis of strongly connected directed networks controlled by the simple synchronizer, in which transmission delays, as well as the time it takes a processor to complete a computational step are random variables. Our main performance measure is the *rate* of computation R_v, i.e., the average number of computational steps executed by a processor in the network, per unit time. To facilitate the presentation, we first assume that the transmission delays are negligible, and only at the end of the paper describe how to extend the results for networks with non-negligible delays.

In Section 3 we study the case in which the random variables have general probability distributions. We consider two approaches. First (Section 3.1) we analyze the effect of the topology on the rate. We use stochastic comparison techniques to compare the rate of networks with different topologies. We give examples of networks with different topologies, but with the same rate. Then (Section 3.2) we analyze networks with the same topology but different processing times. By defining a partial order on the set of distributions, we show that deterministic (i.e. constant) processing times maximize the rate of computation. For this case, it is shown in [ER1] that if the processing times are equal to λ^{-1}, the rate of the network is λ, regardless of the number of processors in the network or its topology. In the next section we show that in case the processing times are random and unbounded, the rate may be degraded by a logarithmic factor in the number of processors. This occurs in the case of exponentially distributed processing times. However, in this section we show that the exponential is the worst, among a large and natural class of distributions (it yields the minimum rate within a class of distributions).

In Section 4 we concentrate on the case of processing times that are exponentially distributed random variables with mean λ^{-1}. We prove that the rate is between $\lambda/4\log(\Delta+1)$ and $\lambda/\log(\delta+1)$, where Δ (δ) is the maximum (minimum) vertex in-degree or out-degree. Hence, for regular-degree (either in or out-degree) networks, the rate is $\Theta(\lambda/\log(\delta+1))$. We compute the exact rate and the stationary probabilities for the extreme cases of a directed cycle and a complete graph. Finally, we study the effect of having one processor that runs slower than the rest of the processors, and we show that in some sense, the directed cycle network is more sensitive to such a bottleneck processor than a complete network.

In the last section we show that it is easy to extend the results to networks with non-negligible transmission delays. We consider the exponential distribution case, and show that adding transmission delays to a regular degree network may reduce its rate by at most a constant factor, provided that they are not larger (w.r.t. the partial order) than the processing times. In networks with processing times exponentially distributed with mean 1, and larger delays with mean λ^{-1}, we compare the results with those of [ER2], where it was shown that for the corresponding deterministic case the rate is λ. In the probabilistic case of a regular-degree network, the rate is at least $\Theta(\lambda/\log\delta)$. Thus, in both cases (small and large delays), the rate of a bounded degree network is reduced only by a constant factor.

Previous Work

There exist several results related to our results in Section 3.2 in the literature on stochastic petri-nets. For instance, dominance results for rather general stochastic petri-nets have been obtained in [Ba89] and more recently in [BL91] by using Subadditive Ergodic Theory (e.g. [K73]). It should be noted, however, that the proofs we provide for the simple synchronizer are different and much simpler and do not require heavy mathematical tools. Other stochastic ordering studies exist. Papers on acyclic networks and fork-join queues are [PV89] and [BM89, BMS89, BMT89], respectively. For closed queueing networks the effect of increasing the service rate of a subset of stations for systems such that the distribution of the number of works in each station has a product form solution is studied in [SY86].

A model similar to our model in Section 4 is considered in [BT89], where it is claimed that the rate is $\theta(1/\log \delta_{out})$, for regular networks with out-degree equal to δ_{out}, identically exponentially distributed transmission delays with mean 1, and negligible processing times. In [BS88] only a lower bound of $\Theta(1/\log \delta_{in})$ on the rate is given, for regular networks with in-degree equal to δ_{in} , with negligible transmission delays, and identically exponentially distributed processing times. Recently, it has been shown in [BK91] that subadditive ergodic theory can be used to derive more general lower bounds on the rate. A bottleneck problem related to ours has been considered by [B88] where an asymptotic analysis of cyclic queues as the number of costumers grows is presented. Asymptotic performance of stochastic marked graphs as the number of tokens grows is studied in [M2]. The class of networks with exponentially distributed processing times belongs to the more general model of stochastic petri nets (see [Ma89] for a survey), where it is usually assumed that the state space (of exponential size, in our case) is given.

2 THE MODEL

The *network* is modeled by a (finite) directed, strongly connected graph $G(V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of vertices of the graph and $E \subseteq V \times V$ is the set of directed edges. A vertex of the graph corresponds to a processor that is running its own program, and a directed edge $u \to v$ corresponds to a communication link from processor u to processor v. In this case, we shall say that u is an *in-neighbor* of v, and v is an *out-neighbor* of u in the network. The processors communicate by sending messages along the communication links. To facilitate the presentation, we assume that the message transmission delays are negligible. At the end we briefly discuss the case of non-negligible transmission delays.

Initially, all processors are in a *quiescent state,* in which they send no messages and perform no computations. Once a processor leaves the quiescent state, it never reenters it and is considered *awake*. When awakened, each processor operates in phases as described in the sequel. Assume that at an arbitrary time, $t(v)$, processor v leaves the quiescent state and enters its first *processing state*, PS_0 (this may be caused by a message from another processor, or a signal from the outside world, not considered in our model). Then, processor v remains in

PS_0 for $\tau_0(v)$ units of time and then transits to its first *waiting state*, WS_0. ¿From this time on, let PS_k and WS_k, $k \geq 0$, denote the processing state and the waiting state, respectively, for the k-th phase. Observe that we are concerned with the rate of computation of the network; the nature of the computation is of no concern to us here. Thus we take the liberty of denoting with the same symbol the k-th processing state of all the processors.

The transition rules between states are as follows: If a processor v transits from state PS_k to WS_k, it sends one message on each of its outgoing edges. These messages are denoted by M_k. Note that this labeling is not needed for the implementation of the protocol; it is used only for its analysis. When v sends the M_k messages, we say that v has completed its k-th *processing step.*

If a processor v is in state WS_k, and has received a message (M_k) on each of its incoming edges, it removes one message from each of its incoming edges, transits to state PS_{k+1}, remains there for $\tau_{k+1}(v)$ units of time and then transits to state WS_{k+1}. Otherwise, (if at least on one incoming edge, M_k has not yet arrived) processor v remains in state WS_k until it receives a message from each of its in-neighbors, and then operates as described above.

The *processing times*, $\tau_k(v)$, correspond to the time it takes for processor v to complete the k-th computation step. The processing times $\tau_k(v)$, $k \geq 0$, $v \in V$, are positive, real-valued random variables defined over some probability space.

For $k \geq 0$, let $t_k^G(v)$ (or $t_k(v)$, whenever G is understood) be the k-th completion time, i.e., the time at which processor v sends messages M_k in network G. Let the *in-set* of a vertex v in G, $IN^G(v)$ (or simply $IN(v)$), be the set of vertices in G that have an edge to v, including v itself, that is, $IN(v) = \{u : u \to v \in E\} \cup \{v\}$. With this notation, the operation of processor $v \in V$ is as follows. Once v has sent a message M_k at time $t_k(v)$, it waits until all processors with an edge to it send message M_k, and then starts its $(k+1)$-st computation step; that is, after the maximum of $t_k(u)$, $u \in IN(v)$, it starts the $(k+1)$-st computation step, which takes $\tau_{k+1}(v)$ units of time, and then sends out M_{k+1}. For this reason we shall assume in the rest of the paper that for each vertex v the edge $v \to v$ is in E. The evolution of the network can be described by the following recursions:

$$\begin{aligned} t_0(v) &= t(v) + \tau_0(v) \\ t_{k+1}(v) &= \max_{u \in IN(v)} \{t_k(u)\} + \tau_{k+1}(v), \quad k \geq 0. \end{aligned} \tag{1}$$

It is interesting to note that the completion times $t_k(v)$ have a simple graph theoretic interpretation. For a vertex v, let $S_k(v)$ be the set of all directed paths of length k ending in v. For $k = 0$, the only path of length 0 ending in v consists of v itself. For a path $P_k = v_0 \to v_1 \to \cdots \to v_k (= v)$, let $T(P_k) \triangleq t(v_0) + \sum_{i=0}^k \tau_i(v_i)$, and $T(S_k(v)) \triangleq \{T(P) : P \in S_k(v)\}$. Thus, $T(S_k(v))$ is a set of random variables; each one is the sum of $k+1$ random variables. Note that these random variables are not independent, even if the $\tau_i(v)$'s are independent. The explicit computation of $t_k(v)$ is as follows.

Theorem 2.1 *For every $v \in V$, $k \geq 0$, $t_k(v) = \max T(S_k(v))$.*

Proof: By induction on k. For $k = 0$, note that the only path of length 0 to v, is v itself, i.e. $S_0(v) = \{v\}$, and $T(S_0(v)) = \{t(v_0) + \tau_0(v)\}$. Hence,

$$t_0(v) = \max T(S_0(v)) = t(v) + \tau_0(v).$$

Assume that the Theorem holds for $k \geq 0$. From the recursion above we have that

$$t_{k+1}(v) = \max_{u \in IN(v)} \{t_k(u)\} + \tau_{k+1}(v).$$

By the inductive hypothesis,

$$t_{k+1}(v) = \max_{u \in IN(v)} \{\max T(S_k(u)) + \tau_{k+1}(v)\},$$

which gives the desired result

$$t_{k+1}(v) = \max T(S_{k+1}(v)).$$

■

The Performance Measures

The most important performance measures investigated in this paper are the completion times $t_k(v)$, $k \geq 0$, $v \in V$. A related performance measure of interest is the counting process $N_t^G(v)$ (or simply $N_t(v)$), associated with processor v defined by

$$N_t(v) \triangleq \sup \{k : t_k(v) \leq t\},$$

that is, $N_t(v)$ is the number of computation steps (minus 1) completed by v up to time t, or the highest index of an M_k message that has been sent by v up to time t. Similarly, $N_t \triangleq \sum_{v=1}^{n} N_t(v)$ denotes the total number of processing steps (minus n) executed in the network up to time t. The following claim indicates that no processor can advance (in terms of executed processing steps) too far ahead of any other processor.

Claim 2.2 *Let d be the diameter of a directed, strongly connected graph G. Then for all $u, v \in V$, and $t \geq 0$, $|N_t(u) - N_t(v)| \leq d$.*

Proof: Denote by l the length of a simple path from u to v. A simple inductive argument on l, shows that the fact that the last message sent by u up to time t is $M_{N_t(u)}$, implies that $N_t(v) \leq N_t(u) + l$. Thus, $N_t(v) - N_t(u) \leq l \leq d$. The same argument for a simple path from v to u proves that $N_t(u) - N_t(v) \leq d$. ■

Another important performance measure is the *computation rate*, $R^G(v)$, (or simply $R(v)$) of processor v in network G, defined by

$$R(v) \triangleq \lim_{t\to\infty} \frac{N_t(v)}{t},$$

whenever the limit exists. Similarly, the computation rate of the network is defined by

$$R \triangleq \lim_{t\to\infty} \frac{N_t}{t}.$$

Claim 2.2 implies that for every $u, v \in V$, $R(u) = R(v)$, and therefore $R = n \cdot R(v)$.

3 GENERAL PROBABILITY DISTRIBUTIONS

In this section we compare the performance of different networks, with general distributions of the processing times $\tau_k(v)$. We first show that adding edges to a network with an arbitrary topology slows down the operation of each of the processors in the network. We show how the theory of graph embedding can be used to compare the rates of different networks. As an example we present graphs, which have the same rate (up to a constant factor) for general distributions, although they have different topologies. Finally, we compare networks with the same (arbitrary) topology but different distributions of the processing times. Specifically, we show that determinism maximizes the rate, and exponential distributions minimize the rate, among a large class of distributions.

3.1 Topology of the Network

3.1.1 Monotonicity

Here we show that adding edges to a network with an arbitrary topology slows down the operation of each of the processors in the network. The basic methodology used is the *sample path* comparison; that is, we compare the evolution of message transmissions in different networks for every instance, or realization, of the random variables $\tau_k(v)$. This yields a stochastic ordering between various networks [Ro83], [S84].

Theorem 3.1 *Let $G(V, E)$ be a graph, and $E' \subseteq V \times V$ be a set of directed edges. Let $H(V, E \cup E')$ be the graph obtained from G by adding edges E'. Assume that processor v, $1 \le v \le n$ awakens in both G and H at the same time $t(v)$. For every realization of the random variables $\tau_k(v)$, $k \ge 0$, $1 \le v \le n$, the following inequalities hold*

$$t_k^G(v) \le t_k^H(v),$$

for all $k \ge 0$, $1 \le v \le n$.

Proof: The proof is by induction on k. The basis of the induction is trivial since

$$t_0^G(v) = t(v) + \tau_0(v) = t_0^H(v).$$

The induction hypothesis is $t_k^G(v) \leq t_k^H(v)$. We need to show that $t_{k+1}^G(v) \leq t_{k+1}^H(v)$. ¿From equation (1) we have that

$$t_{k+1}^G(v) = \max_{u \in IN^G(v)} \{t_k^G(u)\} + \tau_{k+1}(v), \quad t_{k+1}^H(v) = \max_{u \in IN^H(v)} \{t_k^H(u)\} + \tau_{k+1}(v). \tag{2}$$

Since $IN^G(v) \subseteq IN^H(v)$, it follows that

$$\max_{u \in IN^G(v)} \{t_k^G(u)\} \leq \max_{u \in IN^H(v)} \{t_k^H(u)\}.$$

and therefore it follows from (2) that $t_{k+1}^G(v) \leq t_{k+1}^H(v)$, for all v. ■

The previous theorem implies immediately

Corollary 3.2 *Under the conditions of Theorem 3.1 we have that* $N_t^G(v) \geq N_t^H(v)$ *and* $R^G(v) \geq R^H(v)$ *(when the limits exist) for all* $v \in V$*. Also* $N_t^G \geq N_t^H$*.*

Remark 1 *Notice that no assumption was made about the random variables* $\tau_k(v)$*. In particular, they need not be independent.*

Remark 2 *The sample path proof above implies that the random variable* N_t^G *is stochastically larger than the random variable* N_t^H*, denoted* $N_t^G \geq_d N_t^H$*, i.e.* $\Pr\{N_t^G \geq \alpha\} \geq \Pr\{N_t^H \geq \alpha\}$ *for all* α*.*

Remark 3 *The above implies that if one starts with a simple, directed cycle (a strongly connected graph with the least number of edges) and successively adds edges, a complete graph is obtained, without ever increasing the rate.*

3.1.2 Embedding

The theory of graph embedding has been used to model the notion of one network simulating another on a general computational task (see for example [R88]). Here we show how the notion of graph embedding can be helpful in comparing the behavior and the rates of different networks controlled by the synchronizer.

An *embedding* of graph G in graph H is specified by a one-to-one *assignment* $\alpha : V_G \rightarrow V_H$ of the nodes of G to the nodes of H, and a *routing* $\rho : E_G \rightarrow \text{Paths}(H)$ of each edge of G along a distinct path in H. The *dilation* of the embedding is the maximum amount that the routing ρ "stretches" any edge of G:

$$\text{dilation}(\alpha, \rho) = \max_{u \rightarrow v \in E_G} \text{length}(\rho(u \rightarrow v)).$$

The dilation is a measure of the delay incurred by the simulation according to the embedding. The following theorem is a generalization of Theorem 3.1.

Theorem 3.3 *Let (α, ρ) be an embedding with dilation D, of a graph $G(V_G, E_G)$ in a graph $H(V_H, E_H)$. Assume $t(v) = t(\alpha(v))$, for all $v \in V_G$, and that $\tau_k(v)$ and $\tau_{kD}(\alpha(v))$ for all $k \geq 0$, $v \in V_G$, have the same distribution. For every realization of the random variables $\tau_k^G(v) = \tau_{kD}^H(\alpha(v))$, $k \geq 0$, $v \in V_G$, the following inequalities hold*

$$t_k^G(v) \leq t_{kD}^H(\alpha(v)), \quad k \geq 0, v \in V_G.$$

Proof: For each path of length $k \geq 0$ in G,

$$P_k^G = v_0 \to v_1 \to \cdots \to v_k (= v),$$

one can use ρ to construct a path in H of length less than or equal to $k \cdot D$ from $\alpha(v_0)$ to $\alpha(v_k)$, passing through $\alpha(v_1), \alpha(v_2), \ldots, \alpha(v_{k-1})$:

$$\rho(v_0 \to v_1) \to \rho(v_1 \to v_2) \to \cdots \to \rho(v_{k-1} \to v_k).$$

Moreover, there is such a path of length exactly $k \cdot D$, since one can revisit vertices (each vertex has a selfloop) each time between a pair of vertices $\alpha(v_i)$, and $\alpha(v_{i+1})$, there are less than D edges in the path $\rho(v_i, v_{i+1})$. Thus, there is a path in H:

$$P_{kD}^H = u_0 \to u_1 \to \cdots \to u_{kD},$$

where $u_{iD} = \alpha(v_i)$, $0 \leq i \leq k$.

We are assuming a realization $\tau_k(u) = \tau_{kD}(\alpha(u))$, for every $u \in V_G$. It follows that for every path P_k^G, there exists a path P_{kD}^H, such that

$$T(P_k^G) \leq T(P_{kD}^H),$$

and thus

$$\max T(S_k^G(v)) \leq \max T(S_{kD}^H(\alpha(v))).$$

By Theorem 2.1, $t_k^G(v) \leq t_{kD}^H(\alpha(v))$. ■

Corollary 3.4 *Under the conditions of Theorem 3.3 we have that $D \cdot N_t^G(v) \geq N_t^H(\alpha(v))$ and $D \cdot R^G(v) \geq R^H(\alpha(v))$ (when the limits exist) for all $v \in V_G$.*

Remarks 1-3 hold in this case too.

A simple corollary of Theorem 3.3 is that if G is a subgraph of H, $N_t^G(v) \geq_d N_t^H(\alpha(v))$. This is because if G is a subgraph of H, then there is an embedding from G in H with dilation 1. In addition, if the number of vertices in G and H are equal, and the dilation of the embedding is D, then G is a *D-spanner* of H (e.g. [PS89], [PU89]), and we have the following.

Corollary 3.5 *If H has a D-spanner G, then $R^G/D \leq R^H \leq R^G$.*

A motivation for the the theory of embedding is simulation. Namely, one expects that if there is an embedding (α, ρ) from G in H with dilation D, then the architecture H can simulate T steps of the architecture G on a general computation in order of $D \cdot T$ steps, by routing messages according to ρ. In our approach, we compare the performance of G and of H under the synchronizer, without using ρ; the embedding is used only for the purpose of proving statements about the performance of the networks. Consider for example the following two results of the theory of embedding [R88].

Proposition 3.6 *For all $n \geq 1$: One can embed the order n Shuffle-Exchange graph in the order n deBruijn graph with dilation 2. One can embed the order n deBruijn graph in the order n Shuffle-Exchange graph with dilation 2.*

Proposition 3.7 *For all $n \geq 1$: One can embed the order n Cube-Connected-Cycles graph in the order n Butterfly graph with dilation 2. One can embed the order n Butterfly graph in the order n Cube-Connected-Cycles graph with dilation 2.*

By Theorem 3.3, the average rate of the graphs of Proposition 3.6 (3.7) are equal up to a constant factor of 2, provided that the processing times of corresponding processors have the same distributions (regardless of what these distributions are).

3.2 Probability Distributions

3.2.1 Deterministic Processing Times

Now we compare networks, say $G(V, E)$ and $H(V, E)$, having the same (arbitrary) topology, but operate with different distributions of the random variables $\tau_k(v)$. To that end, we assume that the processing times $\tau_k^G(v)$, $k \geq 0$, $v \in V$ are independent and have finite mean $\mathrm{E}[\tau_k^G(v)] = \lambda_v^{-1}$.

We say that λ_v is the *potential rate* of v, as this would be the rate of v if it would not have to wait for messages from its in-neighbors. The processing times in H are distributed as in G except for a subset $V' \subseteq V$ of processors, for which the processing times are assumed to be deterministic, i.e. $\tau_k^H(v) = \lambda_v^{-1}, v \in V'$, for $k \geq 0$. We let $\tau_k^H(v) = \tau_k^G(v) = \tau_k(v)$, $k \geq 0$, $v \notin V'$, be any specific realization of the random variables in G. Again, it is assumed that the processors are awakened at the same time in both networks.

Theorem 3.8 *Under the above conditions we have that*

$$t_k^H(v) \leq \mathrm{E}[t_k^G(v)],$$

for all processors v, and $k \geq 0$. The expectation is taken over the respective distributions of processing times of processors of G in V'.

Proof: The proof is by induction on k. For the basis, $k = 0$ observe that

$$\mathrm{E}[t_0^G(v)] = t_0^G(v) = t(v) + \tau_0(v) = t_0^H(v),$$

for $v \notin V'$, and

$$\mathrm{E}[t_0^G(v)] = t(v) + \lambda_v^{-1} = t_0^H(v),$$

for $v \in V'$.

The induction hypothesis is $t_k^H(v) \leq \mathrm{E}[t_k^G(v)]$, and we need to show that $t_{k+1}^H(v) \leq \mathrm{E}[t_{k+1}^G(v)]$, for all $v \in V$.

¿From (1) we have that

$$t_{k+1}^G(v) = \max_{u \in IN(v)} \{t_k^G(u)\} + \tau_{k+1}^G(v),$$

for $v \in V$. Jensen's inequality implies

$$E[t_{k+1}^G(v)] \geq \max_{u \in IN(v)} \{\mathrm{E}[t_k^G(u)]\} + \mathrm{E}[\tau_{k+1}^G(v)].$$

By the induction hypothesis,

$$\mathrm{E}[t_{k+1}^G(v)] \geq \max_{u \in IN(v)} \{t_k^H(u)\} + \mathrm{E}[\tau_{k+1}^G(v)] = t_{k+1}^H(v),$$

since $\mathrm{E}[\tau_{k+1}^G(v)] = \tau_{k+1}(v)$ for $v \notin V'$, and $\mathrm{E}[\tau_{k+1}^G(v)] = \lambda_v^{-1} = \tau_{k+1}^H(v)$, for $v \in V'$. ■

Remark 4 *Theorem 3.8 holds also if the processing times $\tau_k^H(v)$ of processors v of H in V', are deterministic, but not necessarily the same for every k.*

When all processing times in the network H are deterministic, the computation of the network rate is no longer a stochastic problem, but a combinatorial one. Thus, a conclusion of Theorem 3.8 is that in this case, the computation rate of H, obtained via combinatorial techniques ([ER1] and [ER2]), yields an upper bound on the average rate of G. Furthermore, if the times $t_k^H(v)$ are computed, they give a lower bound on $\mathrm{E}[t_k^G(v)]$, for every $k \geq 0$.

3.2.2 More Variable Processing Times

More generally, we study the effect of substituting a random variable in the network (e.g. the processing time of a given processor, for a given computational step) with a given distribution, for a random variable with another distribution on the rate of the network, and define an ordering among probability distributions.

Recall that a function h is *convex* if for all $0 < t < 1$, x_1, x_2, $h(tx_1 + (1-t)x_2) \leq th(x_1) + (1-t)h(x_2)$. A random variable X with distribution F_X is said to be *more variable* than a random variable Y with distribution F_Y, denoted $X \geq_c Y$ or $F_X \geq_c F_Y$, if $\mathrm{E}[h(X)] \geq \mathrm{E}[h(Y)]$ for all increasing convex functions h. The partial order $\leq_c$ is called *convex* order (e.g. [Ro83], [S84]). Intuitively X will be more variable than Y if F_X gives more weight to the extreme values than F_Y; for instance, if $\mathrm{E}[X] = \mathrm{E}[Y]$, then $\mathrm{Var}(X) \geq \mathrm{Var}(Y)$, since $h(x) = x^2$ is a convex function.

Here we compare networks, say $G(V, E)$ and $H(V, E)$ having the same arbitrary topology, but some of the processing times in G are more variable than the corresponding processing times in H, i.e., for some k's and some v's, $\tau_k^G(v) \geq_c \tau_k^H(v)$, while all other processing times have the same distributions in both graphs. When $t^G(v) = t^H(v)$ and all processing times in G (H) are independent of each other, the following holds.

Theorem 3.9 *Under the above conditions the following holds for all processors v, and $k \geq 0$*

$$t_k^H(v) \leq_c t_k^G(v).$$

Proof: ¿From Theorem 2.1 we have

$$t_k(v) = \max\{T(P_k) : P_k \in S_k(v)\},$$

where $P_k = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k (= v)$ is a directed path of length k ending in v, and $T(P_k) = t(v_0) + \sum_{i=0}^{k} \tau_i(v_i)$.

¿From the fact that the τ's are positive and *max* and $\sum$ are convex increasing functions, it follows that $t_k(v)$ is a convex increasing function of its arguments $\{\tau_i(u) : \ 0 \leq i \leq k, u \in P_k, P_k \in S_k(v)\}$. Now we can use Proposition 8.5.4 in [Ro83]:

Proposition 8.5.4: If $X_1, X_2, \ldots, X_n$ are independent r.v., and $Y_1, Y_2, \ldots, Y_n$ are independent r.v., and $X_i \geq_c Y_i$, $i = 1, 2, \ldots, n$, then $g(X_1, X_2, \ldots, X_n) \geq_c g(Y_1, Y_2, \ldots, Y_n)$ for all increasing convex function g which are convex in each of its arguments.

The proof of the theorem now follows since by assumption the τ's in G are independent, the τ's in H are independent, and $\tau_k^H(v) \leq_c \tau_k^G(v)$, $k \geq 0$, $v \in V$. Note that the random variables $T(P_k)$ are not independent. ∎

Corollary 3.10 *Under the above conditions $N_t^G(v) \leq N_t^H(v)$, $R^G(v) \leq R^H(v)$ and $R^G \leq R^H$.*

In the next section we show that if the processing times are independent and have the same exponential distribution with mean λ^{-1}, then the rate of *any* network is at least $\lambda |V| / \log |V|$. We conclude this subsection by characterizing a set of distributions for which the same lower bound holds.

Assume that the expected time until a processor finishes a processing step given that it has already been working on that step for α time units is less or equal to the original expected processing time for that step. Namely, we assume that the distributions of the processing times $\tau_k(v)$, for all $v \in V$, $k \geq 0$, are *new better than used in expectation* (NBUE) (e.g. [Ro83], [S84]), so that if τ is a processing time, then

$$\mathrm{E}[\tau - a | \tau > a] \leq \mathrm{E}[\tau] \ , \qquad \forall a \geq 0.$$

Let $G_d(V, E)$ be a network with deterministic processing times, let $G_e(V, E)$ be a network with corresponding processing times with the same mean, but independent, exponentially distributed, and let $G(V, E)$ be a network with corresponding processing times with the same mean and independent, but with any NBUE distribution. The following theorem follows from the fact that the deterministic distribution is the minimum, while the exponential distribution is the maximum with respect to the ordering $\leq_c$, among all NBUE distributions [Ro83], [S84].

Theorem 3.11 *For every $v \in V$, $k \geq 0$ it holds that $t_k^{G_d}(v) \leq_c t_k^G(v) \leq_c t_k^{G_e}(v)$.*

Some examples of distributions which are less variable than the exponential (with appropriate parameters) are the Gamma, Weibull, Uniform and Normal.

We should conclude this section by pointing out that the interested reader can find similar results for rather general stochastic petri-nets in [Ba89] and [BL91].

4 EXPONENTIAL DISTRIBUTIONS

In this section we assume that the processing times $\tau_k(v)$, $k \geq 0$, $v \in V$ are independent and exponentially distributed with mean λ^{-1}. We first consider general topologies and derive upper and lower bounds on the expected values of $t_k(v)$, and thus obtain upper and lower bounds on the rate of the network. These bounds depend on the in-degrees and out-degrees of processors in the network, but not on the number of processors itself. Then, exploring the Markov chain of the underlying process, we derive the exact rates of two extreme topologies: the directed ring and the fully connected (complete) network. For these two topologies we study also the effect of having a single slower processor within the network.

4.1 Upper and Lower Bounds

Denote by $d_{out}(v)$ ($d_{in}(v)$) the number of edges going out of (into) v in G, and let

$$\Delta_{out} = \max_{v \in V} d_{out}(v) \ , \qquad \Delta_{in} = \max_{v \in V} d_{in}(v);$$

$$\delta_{out} = \min_{v \in V} d_{out}(v) \ , \qquad \delta_{in} = \min_{v \in V} d_{in}(v).$$

Lemma 4.1 (Lower Bound)

(i) *For every $k \geq 0$ there exists a processor $v \in V$ for which*

$$\mathrm{E}[t_k(v)] \geq \max_{u \in V} t(u) + \lambda^{-1}[1 + k \cdot \log \delta_{out}].$$

(ii) *For every $k \geq 0$, and every $v \in V$, the following holds*

$$\mathrm{E}[t_k(v)] \geq \min_{u \in V} t(u) + \lambda^{-1}[1 + k \cdot \log \delta_{in}].$$

Proof: We present a detailed proof for part (i) only; the proof of part (ii) is discussed at the end. We start by proving that for every $k \geq 0$, there exists a (not necessarily simple) path $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$, such that

$$\mathrm{E}[t_{i+1}(v_{i+1})] - \mathrm{E}[t_i(v_i)] \geq \lambda^{-1} \log \delta_{out}, \quad 0 \leq i < k.$$

We assume the statement holds for $k \geq 0$, and prove it for $k+1$. The proof of the basis is identical. Let v_{k+1} be the processor for which the processing time during the $(k+1)$-th computational step is maximum, among the out-neighbors of v_k, i.e.,

$$\tau_{k+1}(v_{k+1}) = \max_{v_k \rightarrow v} \tau_{k+1}(v),$$

Since v_{k+1} will not start the $(k+1)$st computational step before v_k finishes the kth computational step, we have that $t_{k+1}(v_{k+1}) - t_k(v_k) \geq \tau_{k+1}(v_{k+1})$. The quantity $\tau_{k+1}(v_{k+1})$ is equal to the maximum of at least δ_{out} independent and identically distributed exponential random variables with mean λ^{-1}. It is well known (e.g. [BT89], [D70]) that the mean of the maximum of c such random variables is at least $\lambda^{-1} \log c$. It follows that

$$\mathrm{E}[t_{k+1}(v_{k+1})] - E[t_k(v_k)] \geq \lambda^{-1} \log \delta_{out}.$$

We can chose v_0 to be the one with latest waking time $t(v_0)$, and thus $E[t_0(v_0)] = t(v) + \lambda^{-1}$. Therefore, for every $k \geq 0$, there exists a processor v such that

$$\mathrm{E}[t_k(v)] \geq \max_{u \in V} t(v) + \lambda^{-1}[1 + k \cdot \log \delta_{out}].$$

completing the proof of (i). The proof of part (ii) evolves along the same lines, except that we start from v_k and move backwards along a path. ■

Remark 5 *¿From its proof, one can see that Lemma 4.1 holds for any distribution F of the processing times, for which the expected value m_c of the maximum of c independent r.v. with distribution F exists. In this case it implies that $R_v \leq 1/m_c$, with $c = \delta_{out}$ or $c = \delta_{in}$.*

Remark 6 *Lemma 4.1 implies that for the exponential case, the slowdown of the rate is at least logarithmic in the maximum degree of G. By Remark 5, there are distributions (not NBUE, by Theorem 3.11) for which the slowdown is larger; an example is $F(x) = 1 - 1/x^2$, $x \geq 1$, for which the slowdown is at least the square root of the maximum degree of G ([D70] pp. 58).*

Lemma 4.2 (Upper Bound)

(i) *For every $k \geq 1$, for every processor v,*

$$\mathrm{E}[t_{k-1}(v)] \leq \max_{u \in V} t(u) + \frac{4}{\lambda}(1 + k \cdot \log \Delta_{in}).$$

(ii) *For every $k \geq 1$, for every processor v,*

$$\mathrm{E}[t_{k-1}(v)] \leq \max_{u \in V} t(u) + \log |V| + \frac{4}{\lambda}(1 + k \cdot \log \Delta_{out}).$$

Proof: Again we restrict ourselves to the proof of part (i). Recall that Theorem 2.1 states that for every $v \in V$, $k \geq 0$, $t_k(v) = \max T(S_k(v))$. Also, for a path $P_k = v_0 \to v_1 \to \cdots \to v_k$, $T(P_k) = t(v_0) + \sum_{i=0}^{k} \tau_i(v_i)$, but for the moment let $t(v) = 0$, for every v.

By Proposition D.2 of the Appendix,

$$\Pr\left(T(P_{k-1}) \geq \frac{ck}{\lambda} \log \Delta_{in}\right) \leq e^{-\frac{ck}{4} \log \Delta_{in}},$$

for every $c > 4$, since $\log 2 / \log \Delta_{in} \leq 1$. It follows that

$$\Pr\left(t_{k-1}(v) \geq \frac{ck}{\lambda} \cdot \log \Delta_{in}\right) \leq \Delta_{in}^{k} \cdot e^{-\frac{ck}{4} \log \Delta_{in}} = e^{-k(\frac{c}{4}-1) \log \Delta_{in}},$$

for every $c > 4$, and

$$\mathrm{E}[t_{k-1}(v)] \leq \int_0^{\frac{4k}{\lambda} \log \Delta_{in}} 1 dt + \int_{\frac{4k}{\lambda} \log \Delta_{in}}^{\infty} e^{-k(\frac{c}{4}-1) \log \Delta_{in}} \frac{4}{\lambda} \log \Delta_{in} dc = \frac{4k}{\lambda} \log \Delta_{in} + \frac{4}{\lambda}.$$

■

Combining Lemma 4.1 and Lemma 4.2 we obtain:

Theorem 4.3

$$\frac{\lambda}{4 \log \min(\Delta_{out}, \Delta_{in})} \leq R_v \leq \frac{\lambda}{\log \max(\delta_{out}, \delta_{in})}.$$

4.2 Exact Computations

Theorem 4.3 implies the following bounds for the rate of a directed cycle C_n ($\Delta = \delta = 2$) and of a complete graph K_n ($\Delta = \delta = n$), where n is the number of processors:

$$0.36\lambda \leq R^{C_n}(v) \leq \lambda,$$

$$\frac{\lambda}{4\log n} \leq R^{K_n}(v) \leq \frac{\lambda}{\log n}.$$

In this section we shall compute the exact values for the rates of C_n and K_n. To that end we consider the Markov chain associated with the network This Markov chain is denoted by $X(t) = (X_1(t), X_2(t), ..., X_m(t))$, where $X_i(t)$, is the number of messages stored in the buffer of edge i at time t, and m is the number of edges in the network. Note that a processor with positive number of messages on each of its in-coming edges is in a processing state. When such a processor completes its processing (after an exponential time), one message is deleted from each of its in-coming edges and one message is put on each of its out-going edges. We denote by s_0 the state in which $X_i(0) = 1$, $1 \leq i \leq m$. Thus, the network can be represented as a Marked Graph (e.g. [CHEP]).

The number of states in the Markov chain is finite, say N, because a transition of the chain does not change the total number of messages in a circuit in the network. Moreover, if the network is strongly connected, then the Markov chain is irreducible. Therefore, the limiting probabilities P_i, $1 \leq i \leq N$, of the states s_i of the chain exist, they are all positive and their sum is equal to 1 (e.g. [C67],[Ro83]). However, as we shall see, N can be exponential in n, therefore it is infeasible to compute the rate by directly solving the Markov chain. Here we show how to solve the Markov chain for two network classes, without having to produce the entire chain. We hope this combinatorial approach could be applied to other networks as well.

Let G_X denote the transition diagram (directed graph) of the Markov chain X. Consider a *BFS* (breadth first search) tree of G_X, rooted at s_0. The level $L(v)$ of a vertex v will be equal to the distance from s_0 to v. Thus, $L(s_0) = 0$. Denote by L_i, $i \geq 0$, the set of vertices at level i, and by L the number of levels of G_X.

4.2.1 A Simple Directed Cycle

We study the performance of a simple directed cycle of n processors $C_n = p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n \rightarrow p_1$. It is not difficult to observe that the Markov chain associated with C_n, corresponds to that of a closed queuing network; we return to this approach later. Here we choose to use a combinatorial approach.

Theorem 4.4

(i) *All the states associated with C_n, have the same limiting probability.*

(ii) *For any graph G which is not a simple directed cycle (i) does not hold.*

Proof: **(i)** The proof follows from two observations. First, by symmetry, all the states in one level have the same probability. Second, the indegree of any state in the transition diagram is equal to its outdegree. Then a simple inductive argument can be used to prove part (i).

(ii) If G is not a cycle, then it has a node v, s.t. $d_{in}(v) > 1$. Let v_1, v_2, be two nodes with edges to v. Consider the state s, reached from s_0, by the processing completion (or in marked graphs terminology, firing) of vertex v. The outdegree of s is equal to $n-1$, because apart from v, all vertices are still enabled. But the indegree of s is at most $n-2$, because by the firing of v_1, or of v_2 it is not possible to reach s, since there are no messages on the edges from v_1 and v_2 to v, in s. Therefore, we have proved that $d_{in}(s) \neq d_{out}(s)$.

Consider the balance equation that holds at state s: $P_1 + P_2 + ... + P_k = n - 1 \cdot P_s$, Where P_i, $1 \leq i \leq k$ are the limiting probabilities of the states that have an edge to s, $k = d_{in}(s)$, P_s is the limiting probability of s, and $n-1 = d_{out}(s)$. We have just proved that $k \neq n-1$. It follows that it is not possible that all the probabilities of the last equation are equal. ■

The next theorem states that each processor of C_n works at least at half of its potential rate λ, regardless of the value of n.

Theorem 4.5 *The rate $R(v)$ of a processor in C_n is*

$$R(v) = \lambda \cdot \left(1 - \frac{n-1}{2n-1}\right) \sim \frac{\lambda}{2},$$

$$N = \frac{(2n-1)!}{n!(n-1)!}$$

and the limiting probability of each state is $1/N$, where N is the number of states in the associated chain.

Proof: If M is the number of states in which at least one message is in an edge, going into a processor, say v, then the running rate will be M/N times the expected firing rate. This is because v will be enabled when it has more than 0 messages in its input edge, and since all states have the same probability (Theorem 4.4), the percent of the time that is enabled is simply, M/N.

The number of ways of putting n objects in k places is

$$P(n,k) = \frac{(n+k-1)!}{n!(k-1)!}.$$

It is not difficult to see that $N = P(n,n)$ and $M = N - P(n, n-1)$. Thus,

$$\frac{M}{N} = 1 - \frac{n-1}{2n-1}$$

which gives the desired results. ■

4.2.2 A Complete Graph

Let K_n be a complete graph with n processors. Recall that N is the number of states in the associated Markov chain, and let s_0 be the state in which each edge has one token. A state is at level l, $0 \leq l \leq n-1$, if it can be reached from s_0 by the firing of l processors. The limiting probability of a state at level l is denoted by $P(l)$.

Theorem 4.6 *: The rate of a processor in K_n is*

$$R(v) = \lambda / \sum_{i=1}^{n} \frac{1}{i} \sim \frac{\lambda}{\log n}.$$

Proof: A simpler proof can be derived as in the proof of Theorem 4.10; here we give a combinatorial proof which also yields the number and the limiting probabilities of the states of the associated Markov chain.

We consider a Markov chain T, similar to the Markov chain associated with network K_n. The root of T, s_0, is the state with a message in each edge. A state s will have one son for each one of the enabled processors at state s; a son of s corresponds to the state arrived from s by the firing (completion of a processing step) of one of the enabled processors in state s. Note that in chain T there are several vertices corresponding to the same state of the chain associated with K_n.

In T, the number of states in level l is $n!/(n-l)!$, because each time a processor fires it can not fire again until the rest of the processors have fired. Thus, the number N^T, of states in T is

$$N^T = \sum_{i=1}^{n} \frac{n!}{i!}$$

The number of states in which a given processor is enabled at level l, $en(l)$ (edges from level l to level $l+1$), is

$$en(l) = \frac{1}{n} \frac{n!}{(n-l-1)!},$$

because at level l there are $n!/(n-l-1)!$ enabled processors, and by symmetry, each processor is enabled the same number of times at each level.

Let us denote by P_l^T, the limiting probability of a state of T in level l. One can show that $P_l^T = (n-l-1)!/K$, where

$$K = \sum_{l=0}^{n-1} \frac{n!}{(n-l)!} P_l^T = n! \sum_{i=1}^{n} \frac{1}{i}.$$

It follows that the percent of time that a processor is enabled is

$$ut = \sum_{l=0}^{n-1} ut(l) = \frac{1}{\sum_{i=1}^{n} \frac{1}{i}},$$

where $ut(l) = en(l) P_l^T$, and its rate is $\lambda \cdot ut$. ∎

Corollary 4.7 *For a network K_n,*

$$N = 2^n - 1,$$
$$P_l = \frac{l!(n-l-1)!}{n! \sum_{i=1}^{n} \frac{1}{i}}.$$

Proof: As noted before, it may be that two states of T correspond to the same state, say s, of K_n. In fact, if a state of T is reached from s_0 by firing a sequence of processors of length k, then all $k!$ permutations of the processors in this sequence constitute a valid firing sequence, which leads to the same state s. Thus, the limiting probability of a state s at level l is

$$P_l = l! P_l^T = \frac{l!(n-l-1)!}{n! \sum_{i=1}^{n} \frac{1}{i}},$$

The number of different states at level l is $n!/l!(n-l)!$, and the total number of different states is

$$\sum_{l=0}^{n-1} n!/l!(n-l)! = 2^n - 1.$$

■

Corollary 4.8 *Asymptotically, the rate of any network of n processors is between $\lambda n/2$ and $\lambda n/\log n$.*

Observe that the best possible rate of a processor is 2/3 of the potential rate, in the case of a cycle of two processors; adding more processors can only lower this rate, but not below 1/2. Yet, the rate of the network grows linearly with n. In the case of a complete graph, the rate of a processor reduces as n grows, but also here the total number of computational steps executed per unit time ($n/\log n$) grows with n.

4.3 Bottlenecks

Suppose that the potential rate of all processors of a graph is λ, except for one, which has a lower rate μ. We shall now show that such a bottleneck has a stronger effect in a network which is a directed cycle, than in one which is a complete graph.

Consider the case of a simple directed cycle with n vertices CB_n, where $n-1$ processors have rate λ, and one processor has rate μ. Using standard techniques of Queuing Theory, we prove the following.

Theorem 4.9 *The rate of a processor in CB_n is*

$$\mu \left(1 - \frac{\binom{2n-2}{n} \rho^n}{\sum_{i=0}^{n} \binom{n+i-2}{i} \rho^i} \right); \quad \rho = \frac{\lambda}{\mu}.$$

Proof: Let X_i, $1 \leq i \leq n$ be the number of messages in the buffer of the incoming edge to processor i. The total number of messages in the cycle is equal to n. Since this is a closed queueing system, we have that the limiting probability of the system being in state $(x_1, x_2, \ldots, x_n)$ is given by the following product form [Ro]:

$$\Pr(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = K \left(\frac{1}{\mu}\right)^{x_n} \Pi_{j=1}^{n-1} \left(\frac{1}{\lambda}\right)^{x_j} \text{ if } \sum_{j=1}^{n} x_j = n,$$

and is equal to 0 otherwise, where K is a normalization constant that guarantees that the sum of all the above probabilities is equal to 1. Thus, the probability of having l messages on the incoming edge to processor n is

$$\begin{aligned} \Pr(X_n = l) &= \Pr(X_1 + X_2 + \cdots + X_{n-1} = n - l, X_n = l) \\ &= \sum_{S'} K \left(\frac{1}{\mu}\right)^{l} \left(\frac{1}{\lambda}\right)^{n-l} = |S'| K \left(\frac{1}{\lambda}\right)^{n} \rho^l; \quad 0 \leq l \leq n, \end{aligned}$$

where $\rho = \lambda/\mu$ and $S' = \{(x_1, x_2, \ldots, x_{n-1}) : x_j \geq 0, 1 \leq j \leq n-1, \sum_{j=1}^{n-1} x_j = n - l\}$. Hence,

$$\Pr(X_n = l) = K' \begin{pmatrix} 2n - l - 2 \\ n - l \end{pmatrix} \rho^l, \ \ 0 \leq l \leq n,$$

where K' is the normalization factor determined by the condition $\sum_{l=0}^{n} \Pr(X_n = l) = 1$. Finally,

$$\Pr(X_n = l) = \frac{\begin{pmatrix} 2n - l - 2 \\ n - l \end{pmatrix} \rho^l}{\sum_{j=0} n \begin{pmatrix} 2n - j - 2 \\ n - j \end{pmatrix} \rho^j}, \ \ 0 \leq l \leq n.$$

Now, observe that the rate is simply $\mu n[1 - \Pr(X_n = 0)]$, as the rate of the processor is μ while there are messages in the incoming edge to processor n. ■

Several conclusions can be derived from Theorem 4.9. First, observe that the rate of the cycle cannot exceed μn, thus the slow processor bounds the rate of the network. Moreover, for a fixed n and a very slow processor ($\mu \to 0$ or $\rho \to \infty$), the rate of the network is $\mu n[1 - \begin{pmatrix} 2n-2 \\ n \end{pmatrix} \rho^{-n}]$, namely, as ρ increases, the rate approaches its upper bound μn.

Next, we consider the case where the graph is a complete graph KB_n. We continue to assume that the rate of $n-1$ processors is λ and the n-th processor is slower, operating at rate μ. We shall show that, for fixed μ and λ, as the number of processors n grows to infinity, the influence of the slow processor diminishes, and in the limit, the rate of the network is the same as that of a network with all processors running with the same rate λ.

Theorem 4.10 *The rate of a processor in KB_n is at least $\frac{\lambda}{\rho+\log n}$, $\rho = \lambda/\mu$.*

Proof: Suppose that the network is in state s_0 at a given time, and after some time T_1 it returns to that state; then after some time T_2 it returns again, and so on. Then, $\{T_i, i = 1, 2, \ldots\}$ is a sequence of non-negative independent random variables with a common distribution F, and expected value $\mathrm{E}[T_i]$.

Denote by $N(t)$ the number of events (returning to s_0) by time t. The counting process $\{N(t), t \geq 0\}$ is a renewal process. Therefore, with probability 1,

$$\frac{N(t)}{t} \rightarrow \frac{1}{\mathrm{E}[T_i]} \quad \text{as } t \rightarrow \infty.$$

(See, for example [Ro83]). Moreover, since each time the process returns to s_0, each processor of the network has completed exactly one computational step, it follows that the rate of the network is $1/\mathrm{E}[T_i]$. We proceed to bound $\mathrm{E}[T_i]$.

The expected time of T_i, that takes to return to s_0 is of the form:

$$\alpha_j = \frac{1}{(n-1)\lambda+\mu} + \frac{1}{(n-2)\lambda+\mu} + \cdots + \frac{1}{(n-j)\lambda+\mu} + \frac{1}{(n-j)\lambda} + \frac{1}{(n-j-1)\lambda} + \cdots + \frac{1}{\lambda},$$

for some $1 \leq j \leq n$, depending on when the slow processor completes a computational step. If the system leaves s_0 because the slow processor completed a computational step, $\mathrm{E}[T_i]$ is α_1. In general, if the j-th ($1 \leq j \leq n$) processor to complete a computational step, after leaving s_0, is the slow processor, then $\mathrm{E}[T_i]$ is α_j.

The probability of $\mathrm{E}[T_j]$ being equal to α_j, is not necessarily the same for every j, but for the case $\lambda > \mu$, it holds that $\alpha_j < \alpha_{j+1}$. Thus,

$$\alpha_n = \sum_{k=0}^{n-1} \frac{1}{\lambda \cdot k + \mu}$$

gives an upper bound on the time $\mathrm{E}[T]$ that takes to return to s_0, and $1/\alpha_n$, is a lower bound on the rate of a processor in the network.

We have

$$\alpha_n = \sum_{i=1}^{n-1} 1/(\lambda i + \mu) \leq \frac{1}{\mu} + \frac{1}{\lambda}\sum_{i=1}^{n-1}\frac{1}{i} \leq \frac{1}{\mu} + \frac{1}{\lambda}\log n.$$

We see that $\mathrm{E}[T_i] = O(\frac{1}{\mu} + \frac{1}{\lambda}\log n)$ and thus $R(KB_n)$ is at least $\frac{1}{\frac{1}{\mu}+\frac{1}{\lambda}\log n}$. ■

For fixed μ, R_v is $\Theta(\lambda/\log n)$, but observe that the rate of the network cannot exceed μ. However, when the number of processors n increases to infinity, the rate of the network decreases in proportion to $1/\log n$, as if the slower processor is not in the network.

5 NON-NEGLIGIBLE TRANSMISSION DELAYS

We briefly discuss the case of non-negligible transmission delays. In this model the processing times are random, as before, but the transmission delays are also random. Denote the transmission delay of message M_k, $k \geq 0$, along edge $u \to v$, by $\tau_k(u,v)$, and let $\tau_k(u,u) = 0$, for all $u \in V$. It follows that the behavior of the system is described by the recursions

$$t_0(v) = t(v) + \tau_0(v)$$

$$t_{k+1}(v) = \max_{u \in IN(v)} \{t_k(u) + \tau_k(u,v)\} + \tau_{k+1}(v) k \geq 0.$$

Note that this system is not equal to the one of [BT89], in which the processing times are negligible , and the delays non-negligible, with a self-loop in each processor (to model its processing delay).

Let $P_k = v_0 \to v_1 \to \cdots \to v_k (= v)$ be a path of length k. It is easy to see how to modify the definition of $T(P_k)$:

$$T(P_k) \triangleq t(v_0) + \sum_{i=0}^{k-1} [\tau_i(v_i) + \tau_i(v_i, v_{i+1})] + \tau_k(v_k).$$

Thus a theorem similar to Theorem 2.1 holds, and the corresponding results for general distributions follow.

Consider the case in which the processing times, as well as the transmission delays are exponentially distributed, with the same mean, say 1. It is easy to see that Lemma 4.1 still holds, and that Lemma 4.2 holds up to a factor of 2. Namely, by Theorem 3.9, a regular network with non-negligible delays runs at the same rate that the same network, up to a constant factor, provided that the delays are less or equal (in the convex order) than the processing times. In [ER1] we show that for a network with negligible delays and deterministic processing times equal to 1, the rate of any network is equal to 1. Thus, in this case, random processing times degrade the rate by at most a logarithmic factor in the maximum degree of a processor.

Now, consider the case in which all processing times have mean 1, but the delays have mean λ^{-1} greater than 1, both exponentially distributed. The rate in the deterministic case is equal to λ [ER2], and thus, by Theorem 3.3, in our case the rate is at most λ. One can prove (using Proposition D.2), that also in the case of non-negligible delays, the rate is degraded by at most a logarithmic factor in the maximum degree of a processor, with respect to the (optimal) deterministic case, for any NBUE distribution.

As for the exact computations for networks with average processing times and delays exponentially distributed with mean 1, the rate of a simple cycle can be computed using the same tools of Queuing Theory that we used in the case of negligible delays. To compute the rate of a complete network K_n things are not as straightforward; the structure of the Markov process is more complicated, but by the arguments above, we have that the rate is between

$1/8 \log n$ and $1/\log n$. However, using the ideas of embedding, let us show that the rate of K_n, is at least $1/4 \log n$. Let K'_n be a complete network with negligible delays. Construct G, from K'_n by inserting one vertex in each of its edges. By Theorem 3.1 (or also 3.9), the rate of any processor in G is at least

$$\frac{1}{\log(n + n(n-1))} = \frac{1}{2\log n}.$$

One can show that the rate of any processor v in K_n is greater or equal than half the rate of the corresponding processor in G, using using the fact that there is an embedding of K_n into G of dilation 2. Therefore, the rate of v in K_n is at least $1/4 \log n$.

6 CONCLUSIONS

In this paper we have studied the behavior of synchronizers in networks with random transmission delays and processing times. We attempted to present a self-contained, general study of the synchronizer performance, from the view point of distributed algorithms, rather than providing a deep mathematical study of the underlying stochastic process. In particular, we were interested in comparing the behavior of synchronizers with random delays as opposed to the usual approach of analyzing distributed algorithms with bounded delays. Our main conclusion is that if the delays belong to the natural class of NBUE distributions, the rate of the network is only degraded by a small, local (vertex degree) factor.

We presented several properties of the behavior of the synchronizer for general probability distributions, and described techniques useful to compare the rate of the synchronizer running in networks with different topologies.

For exponential distributions we showed that the expected duration of a round of computation depends on the logarithm of a vertex degree, and hence, the rate of computation does not diminishes with the number of processors in the network. We presented techniques to prove upper and lower bounds on the rate, and to obtain exact computations. We hope the combinatorial approach of these techniques, which was applied to rings, complete networks and regular degree networks, will be used in the future to obtain results for other topologies as well.

7 APPENDIX

The following proposition (similar to pp. 672 in [BT89]) is used to prove the lower bounds on the rate of a network.

Proposition 7.1 (D.2) *Let X_i be a sequence of independent exponential random variables with mean λ^{-1}. For every positive integer k and any $c > 4\log 2$,*

$$\Pr\left(\sum_{i=1}^{k} X_i \geq \frac{ck}{\lambda}\right) \leq e^{-\frac{ck}{4}}.$$

Proof: Fix $\beta \in (0, \lambda)$, and let γ be a positive scalar. A direct calculation yields

$$\mathrm{E}\left[e^{\beta(X_i-\gamma)}\right] = \int_0^\infty e^{\beta(x-\gamma)}\lambda e^{-\lambda x} dx = e^{-\beta\gamma}\frac{\lambda}{\lambda-\beta}.$$

In particular, we can choose γ sufficiently large so that $\mathrm{E}\left[e^{\beta(X_i-\gamma)}\right] \leq 1$. If $\beta = \lambda/2$, then $\gamma \geq 2\lambda^{-1}\log 2$ satisfies the last equation . Using the independence of the random variables X_i, we obtain

$$\mathrm{E}\left[e^{\beta\sum_{i=1}^k (X_i-\gamma)}\right] = \Pi_{i=1}^k \mathrm{E}\left[e^{\beta(X_i-\gamma)}\right] \leq 1.$$

Using the Markov inequality, we obtain

$$e^{\beta km}\Pr\left(e^{\beta\sum_{i=1}^k (X_i-\gamma)} > e^{\beta km}\right) \leq 1.$$

This in turn implies that

$$\Pr\left(\sum_{i=1}^k X_i \geq mk + \gamma k\right) \leq e^{-\beta mk},$$

$$\Pr\left(\sum_{i=1}^k X_i \geq k(m+\gamma)\right) \leq e^{-\beta mk},$$

Let $c = 2m$ and $c_0 = 2\gamma$. Then if $c \geq c_0$, we have $m \geq \gamma$ and

$$\Pr\left(\sum_{i=1}^k X_i \geq kc\right) = \Pr\left(\sum_{i=1}^k X_i \geq k2m\right) \leq \Pr\left(\sum_{i=1}^k X_i \geq k(m+\gamma)\right)$$

$$\leq e^{-\beta mk} = e^{-\alpha ck},$$

where $\alpha = \beta/2$. For our choose of β, we have $\alpha = \lambda/4$, $c_0 = 4\lambda^{-1}\log 2$. ■

Acknowledgments

We would like to thank Gurdip Singh and Gil Sideman for helpful comments.

References

[A1] B. Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM*, Vol. 32, No. 4, pp. 804-823, Oct. 1985.

[A2] B. Awerbuch, "Reducing Complexities of Distributed Max-Flow and Breadth-First-Search Algorithms by Means of Network Synchronization," *Networks*, Vol. 15, 1985, pp. 425-437.

[AP90] B. Awerbuch and D. Peleg, "Network Synchronization with Polylogarithmic Overhead," *Proc. 31st IEEE FOCS*, 1990.

[B88] O.J. Boxma, "Sojourn Times in Cyclic Queues - the Influence of the Slowest Server," in *Computer Performance and Reliability*, G. Iazeolla, P.J. Courtois, O.J. Boxma (Ed.), Elsevier Science Publ. (North-Holland), 1988.

[Ba89] F. Baccelli, "Ergodic Theory of Stochastic Petri Networks," Rapport INRIA No. 1037, 1989.

[BG89] V.C. Barbosa and E. Gafni, "Concurrency in Heavily Loaded Neighborhood-Constrained Systems," *ACM Trans. on Programming Languages and Systems*, Vol. 11, No. 4, pp. 562-584, Oct. 1989.

[BK91] F. Baccelli, P. Konstantopoulos, "Estimates of Cycle Times in Stochastic Petri Nets," Proc. Rutgers Conference on Stochastic Analysis, I. Karatzas (Ed.), Springer-Verlag, 1991.

[BL91] F. Baccelli, Z. Liu, "Comparison Properties of Stochastic Decision Free Petri Nets," INRIA Research Raport No. 1433, May 1991.

[BM89] F. Baccelli and A.M. Makowski, "Queueing Models for Systems with Synchronization Constraints," *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 138-161, January 1989.

[BMS89] F. Baccelli, A.M. Makowski and A. Shwartz, "The Fork-Join Queue and Related Systems with Synchronization Constraints: Stochastic Ordering and Computable Bounds," *Advances in Applied Probability*, Vol. 21, No. 3, pp. 629-660, September 1989.

[BMT89] F. Baccelli, W.A. Massey and A. Towsley, "Acyclic Fork-Join Queueing Networks," *Journal of the ACM*, Vol. 36, No. 3, pp. 615-642, July 1989.

[BS88] P. Berman and J. Simon, "Investigations of Fault-Tolerant Networks of Computers," *Proc. of the 20th ACM STOC*, 1988.

[BT89] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, N.J. 1989.

[C67] K. L. Chung, *Markov Chains With Stationary Transition Probabilities*, Springer-Verlag, 2nd edition, 1967.

[CHEP] F. Commoner, A.W. Holt, S. Even and A. Pnueli, "Marked Directed Graphs," *J. of Computer and System Sciences*, Vol., 5, No 5, Oct. 1971.

[CL85] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. on Computer Systems*, Vol. 3, No 1, Feb. 1985.

[D70] H.A. David, *Order Statistics*, John Wiley & Sons, 1970.

[ER1] S. Even and S. Rajsbaum, "Lack of Global Clock Does Not Slow Down the Computation in Distributed Networks," TR #522, Department of Computer Science, Technion, Haifa, Israel, Oct. 1988. The first part of this paper appears with the title "Unison in Distributed Networks" in *Sequences: Combinatorica, Compression, Security, and Transmission*, R.M. Capocelli (ed.), Springer-Verlag, 1990.

[ER2] S. Even and S. Rajsbaum, "The Use of a Synchronizer Yields Maximum Rate in Distributed Networks," *Proc. of the 22nd ACM STOC*, 1990.

[GRST92] J. Garofalakis, S. Rajsbaum, P. Spirakis, B. Tampakas, "Tentative and Definite Distributed Computations: An Optimistic Approach to Network Synchronization," in *Lecture Notes in Computer Science* 647, Springer-Verlag: Proc. of the 6th Int. Workshop on Distributed Algorithms, Haifa, Israel, November 1992.

[K73] J.F.C. Kingman, "Subadditive Ergodic Theory," Ann. Prob., Vol. 1, No. 6, pp. 883-909, 1973.

[Ma89] M.A. Marsan, "Stochastic Petri Nets: An Elementary Introduction," in *Advances in Petri Nets 1989*, Lecture Notes in CS 424, Springer-Verlag, pp. 1-29, 1989.

[MMZ88] J. Malka, S. Moran and S. Zaks, "Analysis of a Distributed Scheduler for Communication Networks," TR-495, Department of Computer Science, Technion, Haifa, Israel, Feb. 1988. Also in *Lecture Notes on CS* 319, pp. 351-360, Springer Verlag, 1988.

[M1] M. K. Molloy, "Performance Analysis Using Stochastic Petri Nets," *IEEE Trans. on Computers*, Vol. c-31, No. 9, pp. 913-917, Sep. 1982.

[M2] M. K. Molloy, "Fast Bounds for Stochastic Petri Nets," International Workshop on Timed Petri Nets, Torino, Italy, pp. 244-249, July 1985.

[OG87] Y. Ofek and I. Gopal, "Generating a Global Clock in a Distributed System," IBM Research Report, 1987.

[PS89] D. Peleg and A. A. Schaffer, "Graph Spanners," *J. of Graph Theory*, Vol. 13, No. 1, pp. 99-116, 1989.

[PU89] D. Peleg and J. D. Ullman, "An Optimal Synchronizer for the Hypercube," *SIAM J. Computing*, Vol. 18, No. 4, pp. 740-747, August 1989.

[PV89] N. Pekergin and J-M. Vincent, "Stochastic Bounds on Execution Times of Task Graphs," Report EHEI, 1989.

[R88] A. L. Rosenberg, "Shuffle-Oriented Interconnection Networks," COINS Technical Report 88-84, Univ. of Massachusetts, 1988.

[Ra91] S. Rajsbaum, "Stochastic Marked Graphs," 5th Int. Workshop on Petri Nets and Performance Modeling, IEEE Computer Society, Melbourne, 1991, pp. 95-101.

[RM92] Y. Malka, S. Rajsbaum, "Analysis of Distributed Algorithms based on Recurrence Relations," in *Lecture Notes in Computer Science* 579, pp. 242–253, Springer Verlag, 1992: Proc. of 5th Int. Workshop on Distributed Algorithms, Delphi, Greece.

[Ro83] S. M. Ross, *Stochastic Processes*, J. Wiley, 1983.

[S84] D. Stoyan, *Comparison Methods for Queues and Other Stochastic Models*, English Translation (D.J. Daley, Ed.), J. Wiley & Sons, New York, 1984.

[SY86] J.G. Shanthikumar, D.D. Yao, "The effect of Increasing Service Rates in a Closed Queueing Network," *J. Appl. Prob.,* 23, 474–483, 1986.