Availability and Performability

Coverage

- Definitions
- State Models
- Reliability
  - Block Diagrams
  - Success Diagrams
  - Fault Trees
- Availability
- Performability
  - Reward Rates
  - Representations
  - Examples

Availability and Performability

Definitions

Reliability

R(t) = Probability that a system remains operations at time t if it was started in perfect condition at t =0.

Availability

Probability that a system is available at any given time assuming a failure distribution and a repair time distribution.

Performability

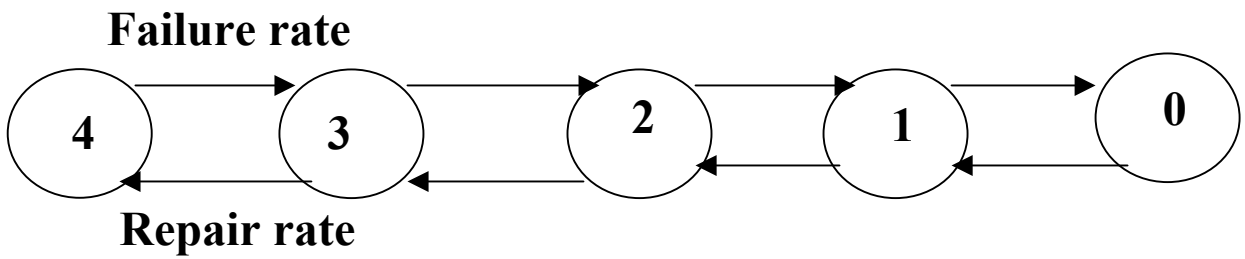
Probability that a service is available at a given level of performance at a given time.

Each measure is the fraction of time that a system is in some state or set of states over a period of time long with respect to a service time.

Each can be computed from a state model where each state is a configuration of the system.

Direct Formulation in Terms of State Models

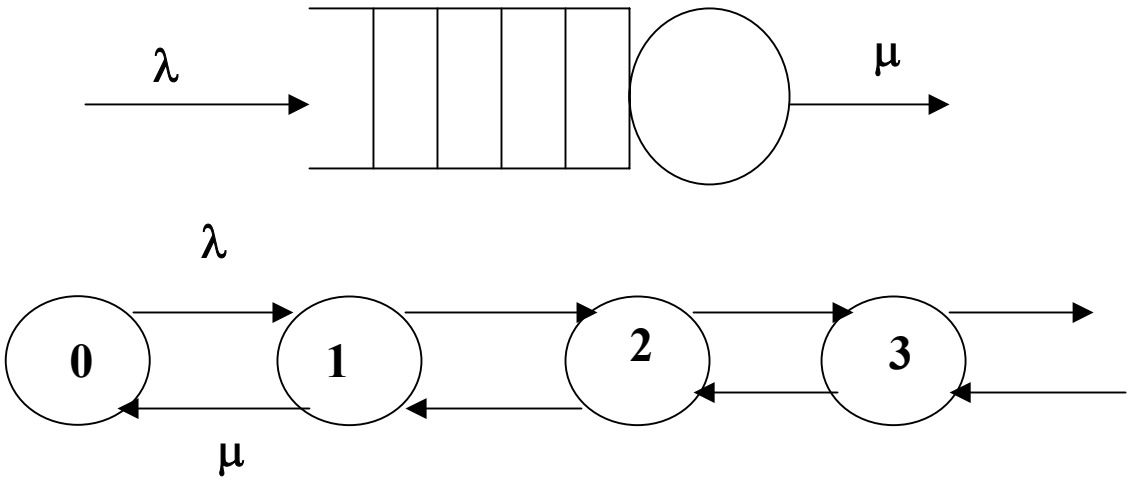
Each of these properties can be formulated by constructing an appropriate state model. Consider a system with up to four processors where two processors are required to implement the service.



The failure rate is the rate at which the system fails and the repair rate is the rate at which a server can be brought back into service after failing. Availability (and in this case performability) is proportional to the probability that there are two or more servers available.

Queueing networks, Stochastic Petri nets and similar representation systems are often more convenient for system specification and property specification. There are also representations closely related to Petri nets which are used for formulation and evaluation of availability and performability models.

States of a Queue/Server Pair



Solution of queuing networks is determining the fraction of time that a queuing network is in a given state.

Therefore performance and reliability can be formulated in similar ways.

Reliability and Its Computation

R(t) = Probability that a system remains operational at time t if it was started in perfect condition at t =0.

Let T be the time of the first failure and let F(t) be the distribution function for the occurrence of the first failure.

Let $\lambda(t)$ be the failure rate.

$$\lambda(t) = (-1/R(t))\,(dR/dt)$$

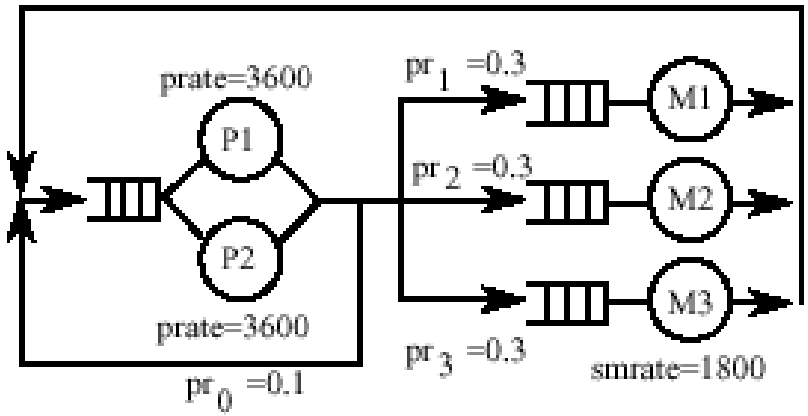
If T is exponentially distributed the $\lambda(t)$ is independent of t and is the rate parameter of the exponential distribution.

If F(t) =is a Weibull distribution then $\lambda(t) = \alpha\beta\,(\beta\, t)^{**}(\alpha-1)$

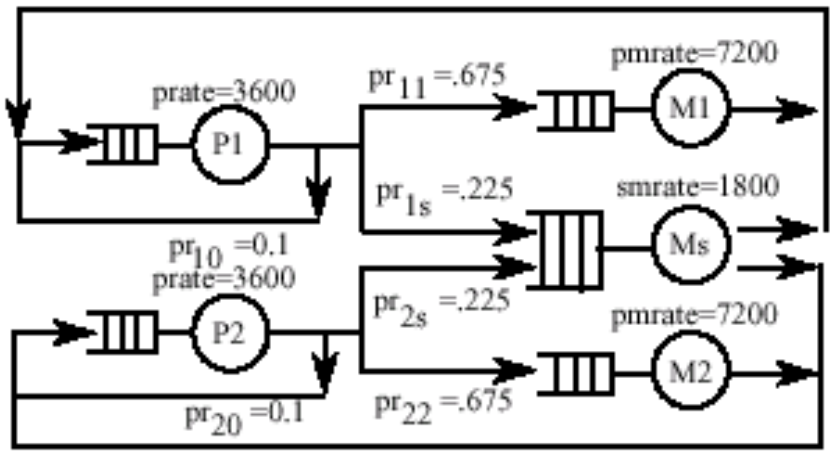
The median of F(t) is the Mean Time To Failure (MTTF).

There is a similar argument leading to Mean Time To Repair (MTTR).

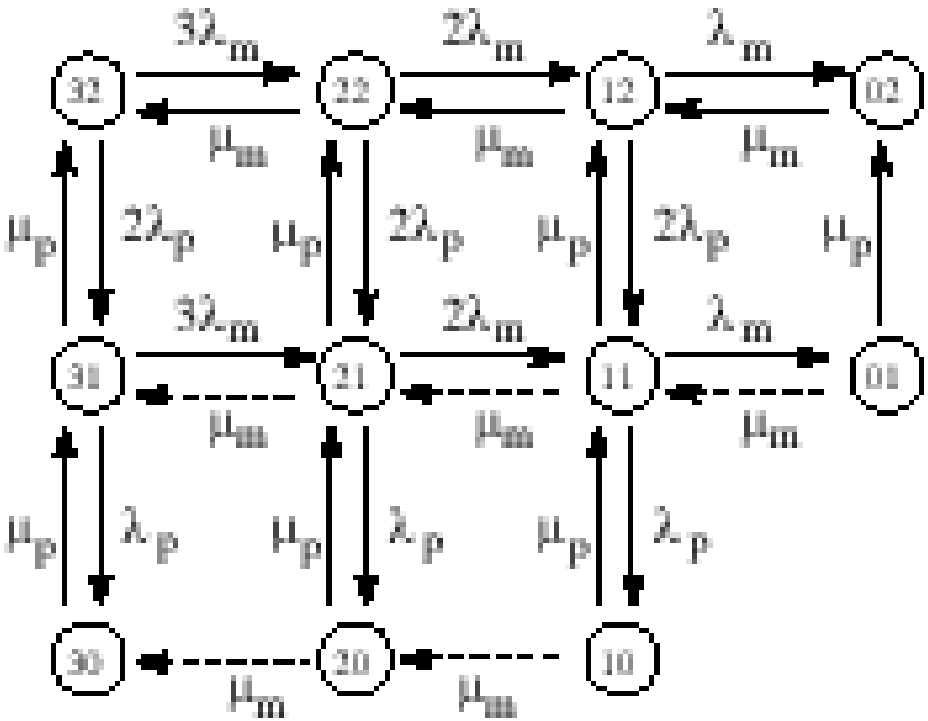
Two-Processor, Three Shared Memories Example - Queuing Network Representation



Two Processors, Three Memories with One Shared Memory



Continuous Time Markov State Model (CTMC) for the two-processor three-memory system. λ_m is the failure rate for a memory, etc. The probability that the system is in a state (mp) is determined from the state diagram below and an initial condition.



Reliability Block Diagrams

A Reliability Block Diagram (RBD) requires that the system be formulated as a network of components where each network segment is either in series or in parallel.

Reliability of components in series

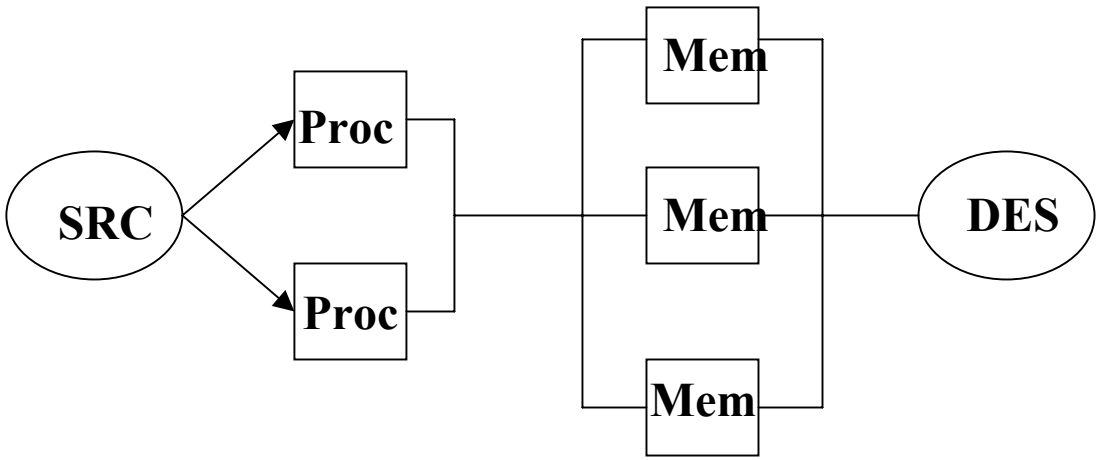
$$R(t) = \Pi R_i (t) \quad (1 <= i <= k)$$

Reliability of components in parallel

$$R(t) = 1 - \Pi (1-R_i(t)) \quad (1 <= i <= k)$$

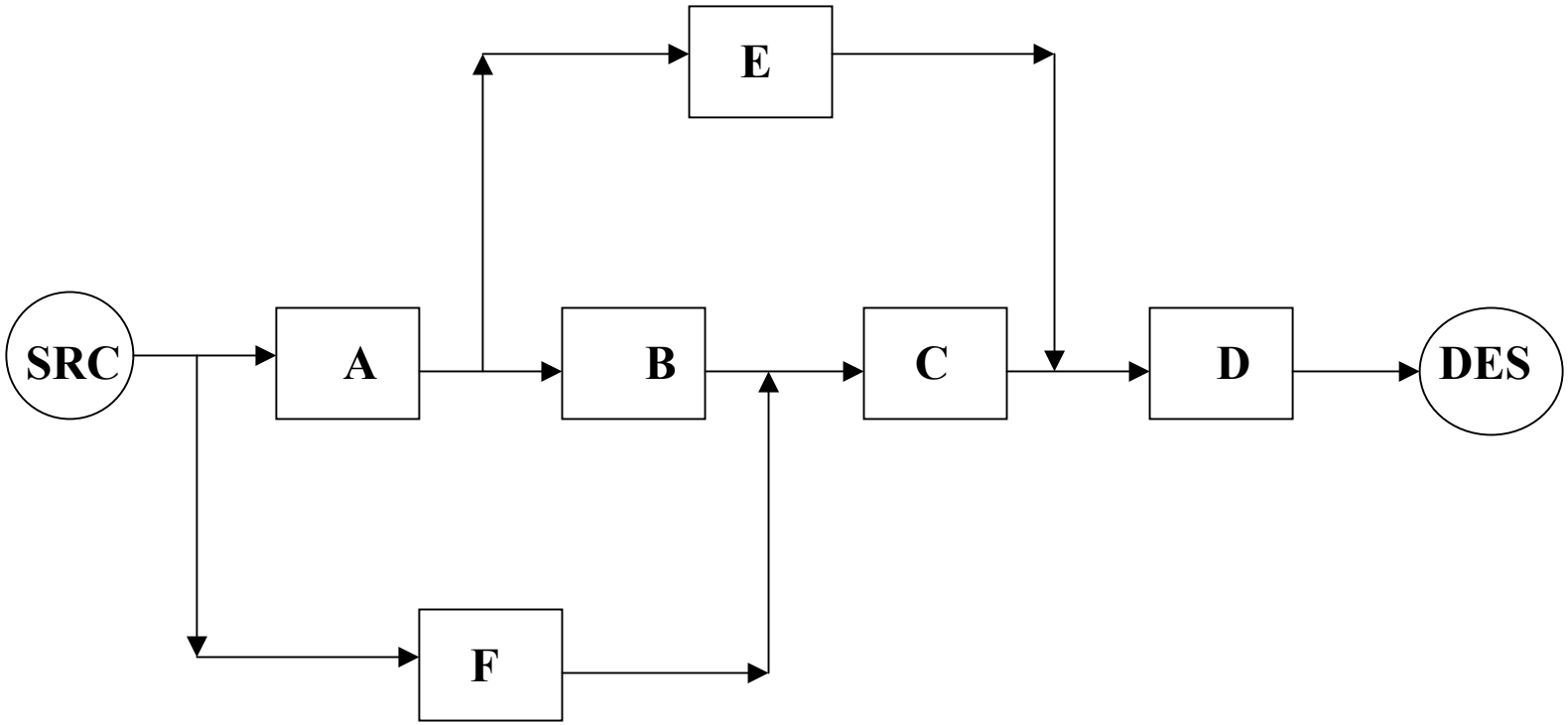
In general one must manipulate the structure of the system or consider special cases to formulate RBDs for arbitrary systems.

Let us look at the simple case where two processors share access to three memories. The system is available as long there is one processor and one memory available.



R(sys) = (1- R(Proc)2)*(1-R(Mem)**3)**

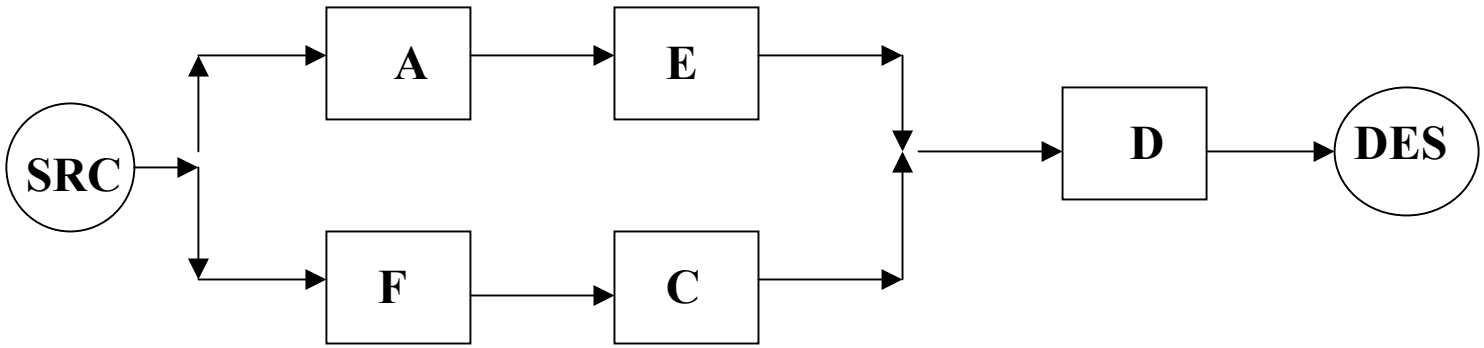
Availability and Performability



Service is available if a path exists from SRC to DES.

To apply RBD methods you must condition the success or failure on the availability of a given component in order to get a system with serial/parallel structure.

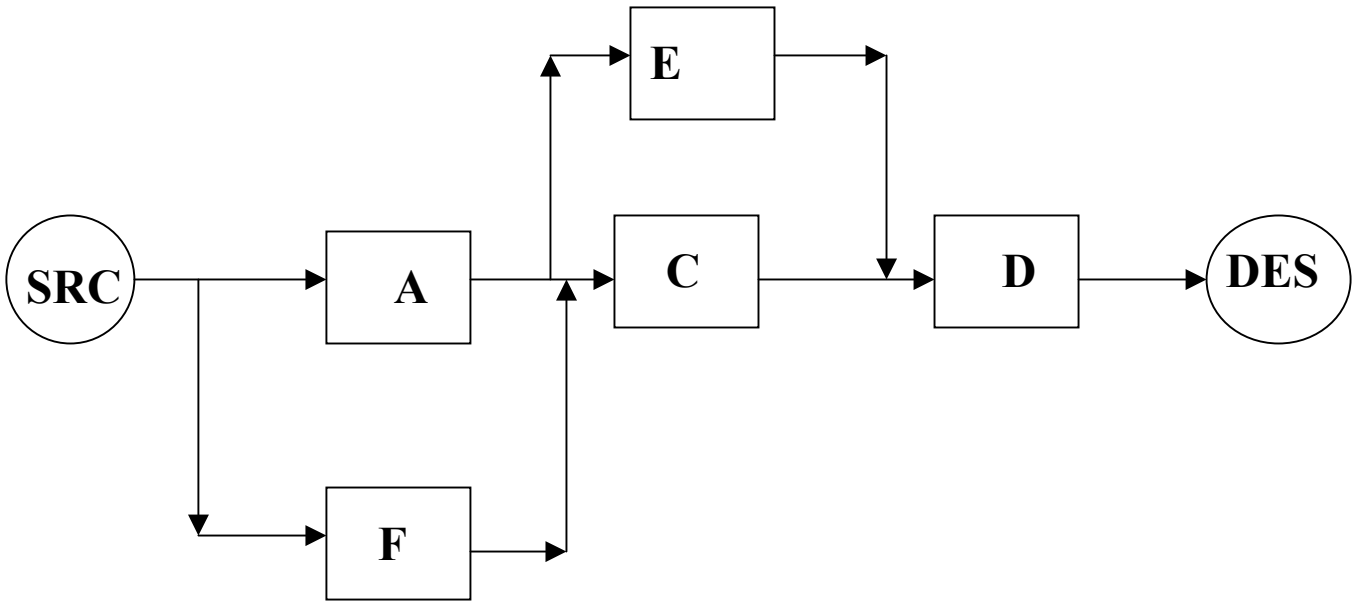
1. Assume that B fails. The resulting network has the following configuration and still works. Call this network R(1).



The contribution of this configuration to system availability is:

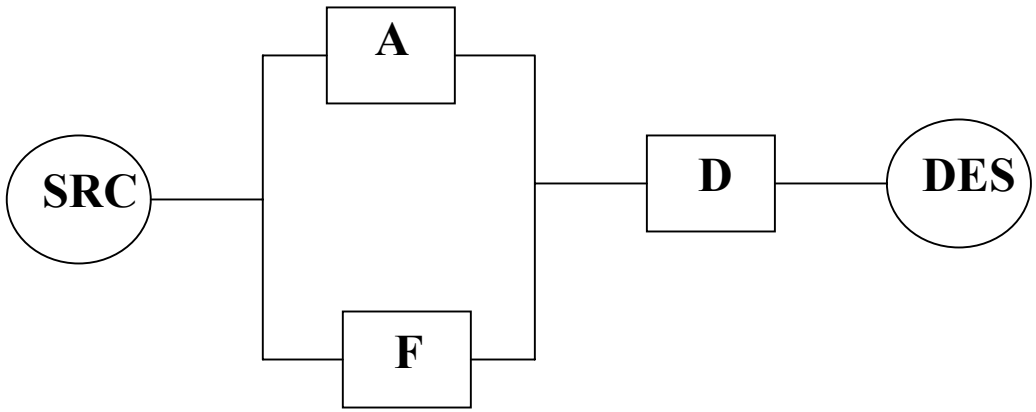
R(sys) = (1-R(B)) R(1)

Now let us assume that B is always in service. Then the network configuration can be written as:



This network configuration is still not series/parallel so we must condition it on C to get two more series/parallel networks.

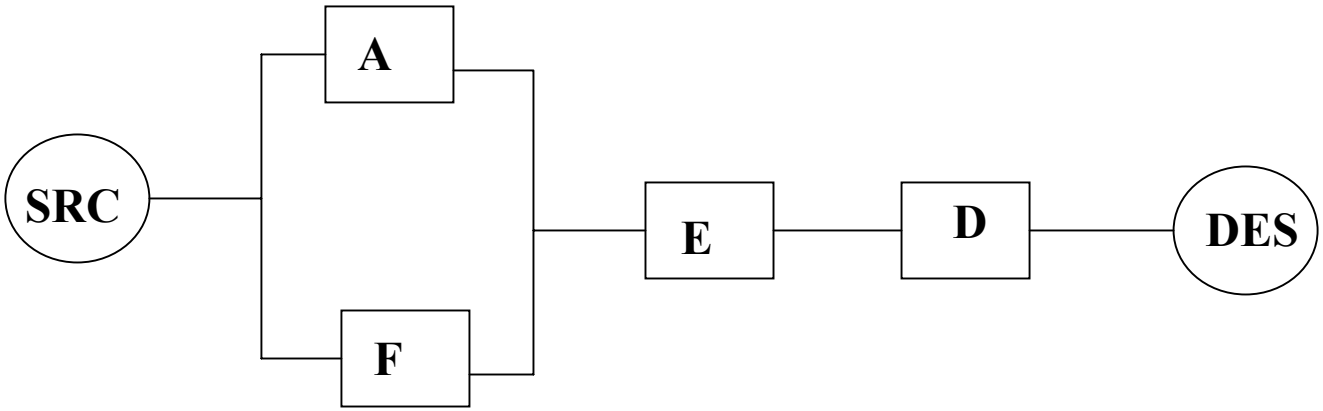
To get a series/parallel system structure assume that C also works all of the time. Then the system no longer depends on the availability of E and we get the system, R(2) below.



The contribution of this structure to system reliability is:

$$R(2) = R(B)R(C)R(2)$$

Let us now assume that B works all of the time but C has failed. Then the failure of E must be taken into account. The system, R(3) below represents this circumstance.



The contribution of this configuration to system reliability is:

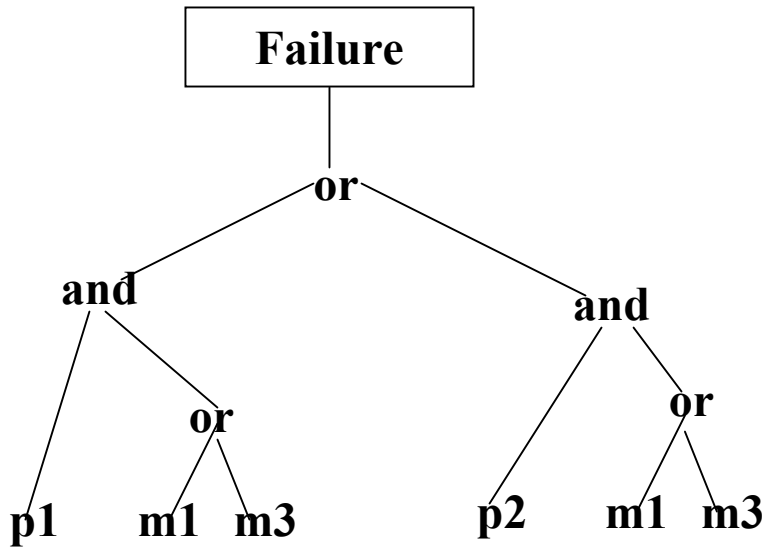
$$R = R(B)(1-R(C)R(3).$$

Thus the total system reliability is:

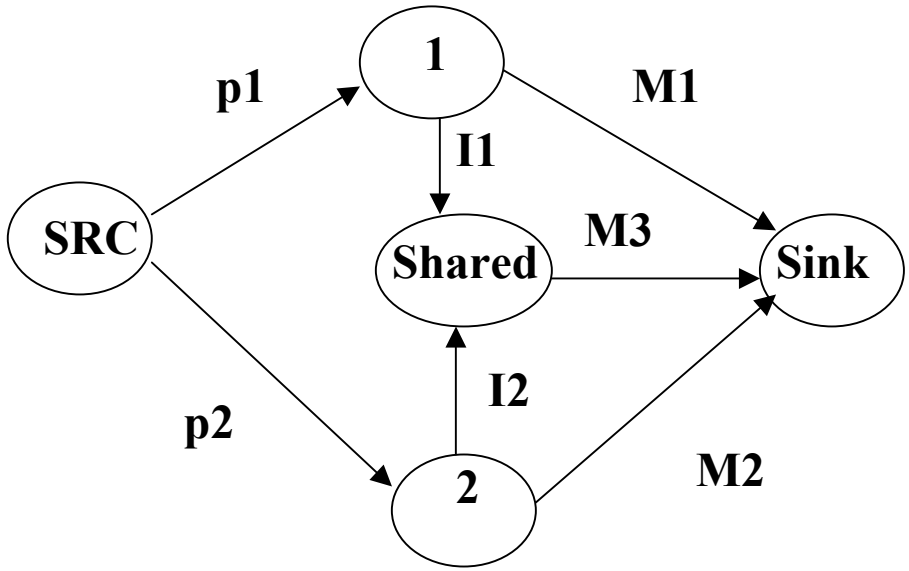
$$R(sys) = (1-R(B))R(1) + R(B)(R(C)R(2) +(1-R(C))R(3))$$

Fault Trees and Reliability Diagrams

Consider a system with two processors, each of which has a fast private memory and access to a slower shared memory. The system is available if at least one processor has access to at least one memory. This cannot be modeled as a series/parallel system. But it can be modeled as a tree.



Reliability Graph of the Same System



Now the resources are the arcs and the system is available as long as there is a path from SRC to Sink. The arcs have the probability of being broken equal to the probability of failure of the component. I1 and I2 are special arcs with zero probability of failure.

Adding Recovery/Repair to Availability Models

1.Let us now "repair" to the system model. Let λ be the failure rate and μ be the repair rate.

2. Let us also assume exponential distributions of failure and repair.

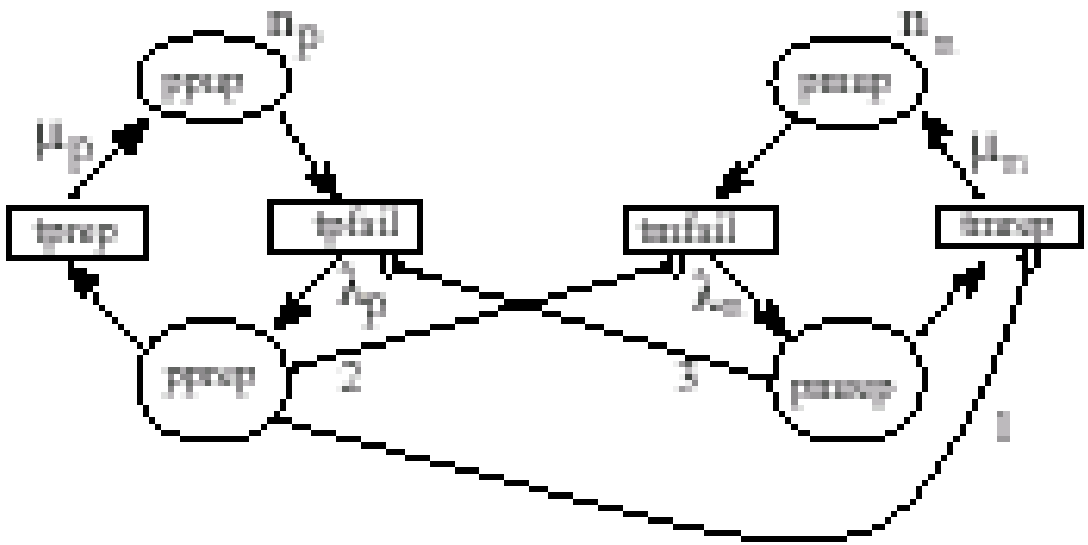
3. Let also assume that any number of repairs can be in progress simultaneously.

4. Using this third assumption we can compute availability by replacing the failure distribution for each component by the availability/unavailability distribution for each component.

5. But if the third assumption is not valid then we must use Markov state models.

GSPN Models of Availability

The availability of this system can also be modeled in GSPN as below. The system is available whenever there is at least one token in place ppup and one token in place pmup. This model assumes there is only one repair facility. This is modeled by the inhibitor arc from pmrep to tpfail.



Performability

Probability that a service is available at a given level of performance at a given time during a time period long with respect to a service time.

Performability Metrics

1. Probability that a system is performing at rate r at time t => r(t).

2. Probability that a system has accomplished a given amount of work in the interval (0,t)

3. Capacity - average rate at which work is accomplished during the interval (0,t).

Computation of Performability Metrics

1. Associate a reward rate, r(t) with each state of the Markov state space.

2. Compute the probability of being in each state of interest.

Stochastic Reward Nets and Stochastic Activity Nets

Stochastic Reward Nets (SRN) and Stochastic Activity Nets (SAN) are two extensions of Stochastic Petri nets which are popular representations for representing systems for the purpose of computing performability metrics.

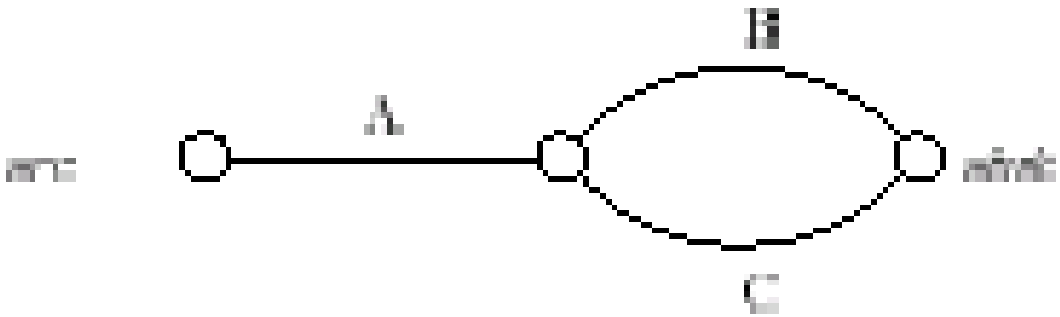
Each associates a reward rate with each state and provides means of computing the reward rates for a system configuration in equilibrium. SAN's, being specifically designed to capture performability give more compact models.

SANs have activities (=transitions), places, input gates and output gates. Input gates have an enabling predicate and a function which changes the marking of the network. Output gates have a function which changes the marking of the network.

We look at a model from Trivedi and a model from Meyers.

1. The simple network below is operational as long as link A is available and one of link B or C is avavailable.

2. Assume each of the links has a separate repair facility.



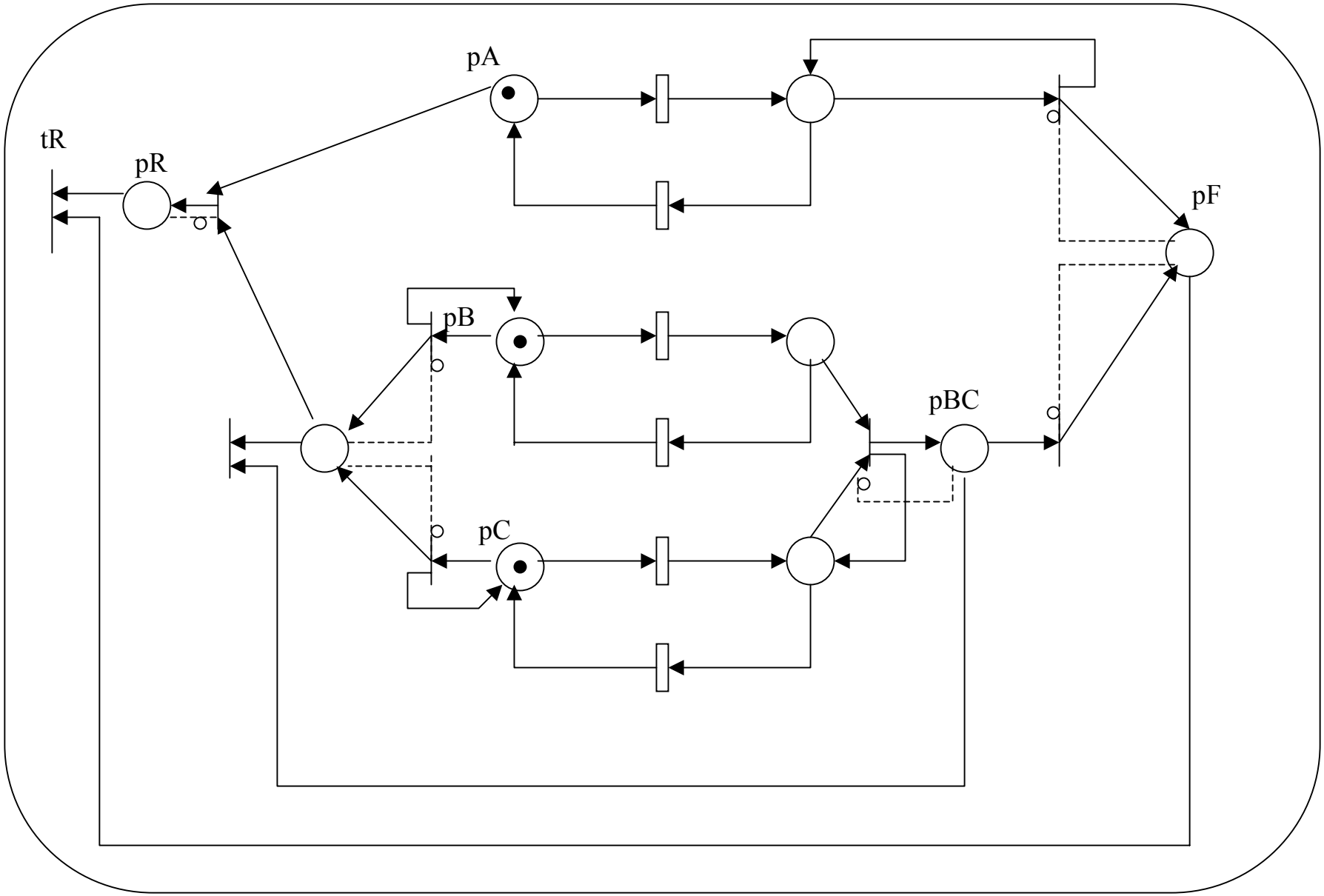
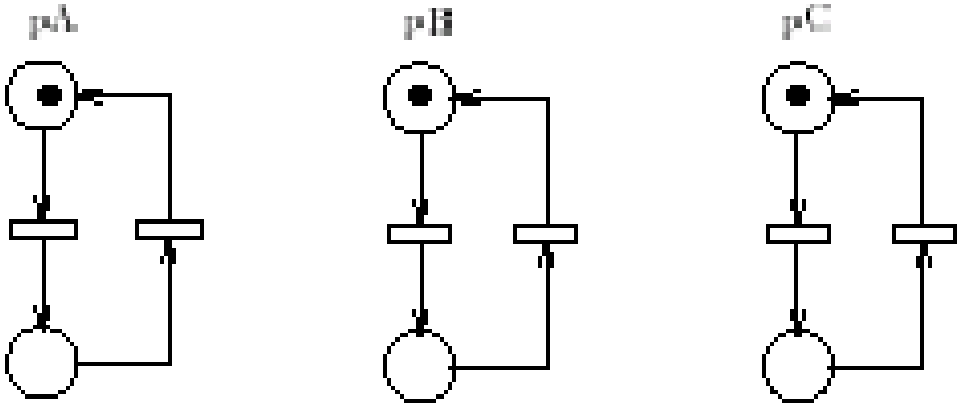


Figure 9: GSPN availability model of the network

Availability and Performability



Name	Boolean Function
bool_	(#fail(pA) == 1)
bool__	(#fail(pB) == 1) || (#fail(pC) == 1)
bool___	bool_ /\ bool__

Reward Rate Functions
if (bool___ == 1) then r = 1 else r = 0

Figure 10: SRN availability model of the network

Faulty Multiprocessor Example

Consider an object system consisting of N processing elements (PEs) that share a common input queue of finite capacity L. The environment is a serial stream of incoming tasks which arrive as a Poisson process with parameter λ. Whenever queue capacity permits, an arriving task enters the object system; if the queue is full, an arriving task is rejected. Tasks are scheduled to PEs on a FIFO basis, where any available" PE (the meaning of this will be explained momentarily) attempts to access the task at the head of the queue. Just which PE receives the task is assumed to be equally likely (among those PEs which are available).

All PEs have identical structure and behave as follows. A PE, having accessed a task when idle, can process it in exponentially distributed time with mean value 1= 1/μ

Moreover, while busy with a single task, it can access a second task and, in turn, simultaneously complete the processing of both as if they were one, i.e., the two are then processed together at the single-task rate μ.

Accordingly, we regard a PE as being available if it is idle or processing a single task; otherwise it is unavailable, i.e., it is engaged with two tasks and does not attempt to access a third.

In the case of double-task processing, however, all is not perfect. Specifically, there is interference, due to a fault in the design, that results in processing errors.

Fortunately, these are detected at the time both tasks complete, with the likelihood of encountering errors being given by the probabilities:

p1 = the probability that exactly one of the two tasks is processed erroneously

p2 = the probability that both tasks are processed erroneously

where 0 <= p1 + p2 <= 1. If a task completes with erroneous processing (we call this an erroneous completion), it is immediately returned to the PE for reprocessing.

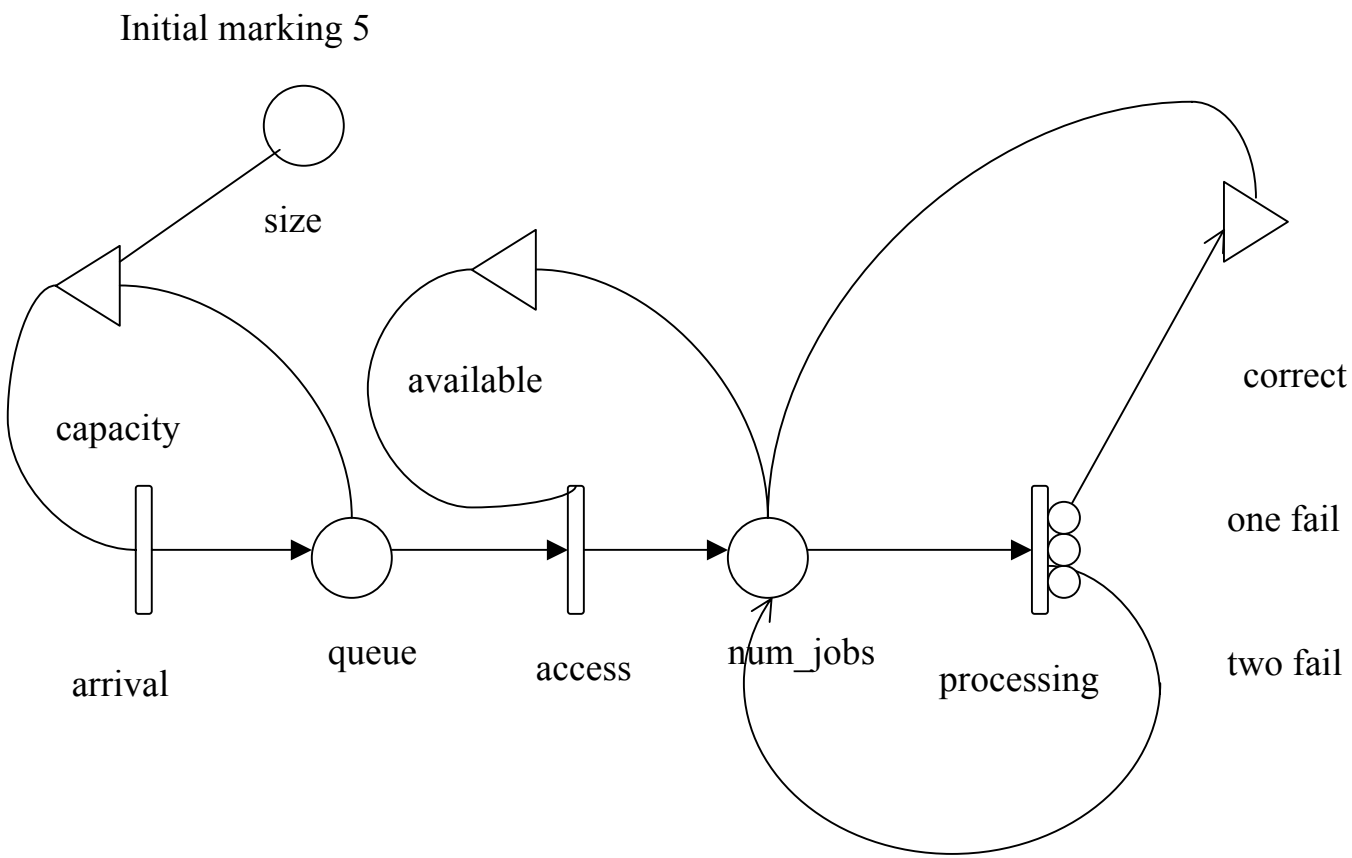
Accordingly, since both jobs may have erroneous completions, we assume further that p2 < 1, so as to eliminate the possibility of live-lock.

When the processing of a task completes without errors (an error-free completion), which may require several processing iterations, it departs the object system.

Thus, any task that enters the system (i.e., is not rejected at the input) will be eventually accessed by some PE and, in turn, will eventually enjoy an error-free completion via that PE.

In the case where only a single task resides in a PE during an iteration (either a task that was just accessed or one that was returned internally for reprocessing) everything works fine and, hence, error-free completion is guaranteed.

Availability and Performability



Capacity and available are input gates and correct is an output gate. The three dots on the activity processing are outcomes.

Table 1: Activity Time Distributions

Activity	*Distribution*	*Parameter values*
access	exponential	
	rate	1000
arrival	exponential	
	rate	1.5
processing	exponential	
	rate	1

Table 2: Activity Case Probabilities

Activity	*Case*	*Probability*
processing	1	*if (MARK(num_jobs) == 1)* *return(1.0);* *else return(0.84);*
	2	*if (MARK(num_jobs) == 1)* *return(ZERO);* *else return(0.16);*
	3	*if (MARK(num_jobs) == 1)* *return(ZERO);* *else return(0.01);*

Table 3: Input Gate Definitions

Gate:	*Definitions*
available	Predicate: *MARK(num_jobs) < 2*
	Function */* do nothing */* ;
capacity	Predicate: */* has the buffer capacity been reached? */* *MARK(queue) < MARK(size)*
	Function */* do nothing */* ;

Table 4: Output Gate Definitions

Gate	Definition
correct	/* complete all jobs at the same time */ MARK(num_jobs) = 0;