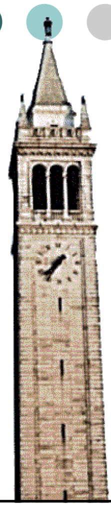


Introduction to Embedded Systems

Edward A. Lee & Sanjit A. Seshia

UC Berkeley

EECS 124

Spring 2008

Lecture 6: Simulation of Discrete-Event Systems

Material drawn from book by Banks et al., notes by M. Harchol-Balter

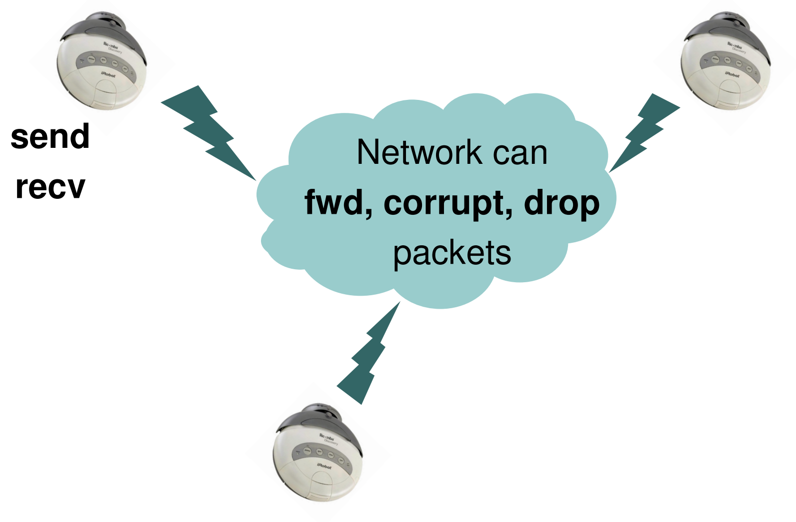
Discrete-Event System

A dynamical system whose evolution is governed by the occurrence of events at discrete time points, at possibly irregularly-spaced intervals (Informal defn)

Many cyber-physical systems are modeled as discrete-event systems:

- Communication networks
- Microprocessors
- Manufacturing facilities
- Communicating robots

Example: Communicating Robots/Sensor Nodes



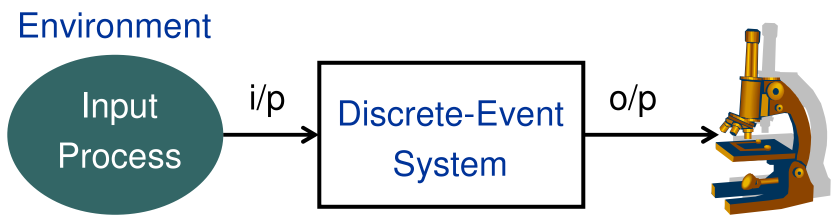
This Lecture

How to build a simulator for a discrete-event system – the basics

Examples of such simulators:

- ns-2 (for simulating computer networks)
- ModelSim (for simulating digital circuit designs)

Simulating a Discrete-Event System (DES)

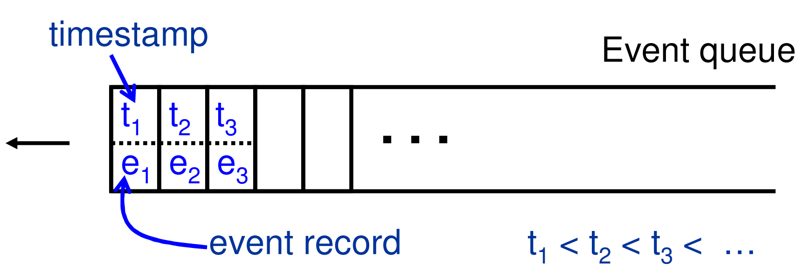


2.
How to generate random input?

1.
How to simulate the system on an input?

3.
How to check Output?

Simulating the System with an Event Queue

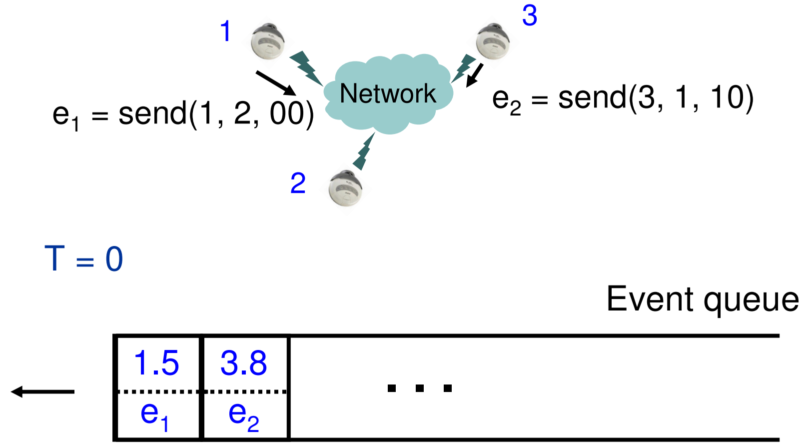


Simulation Timer, T = 0

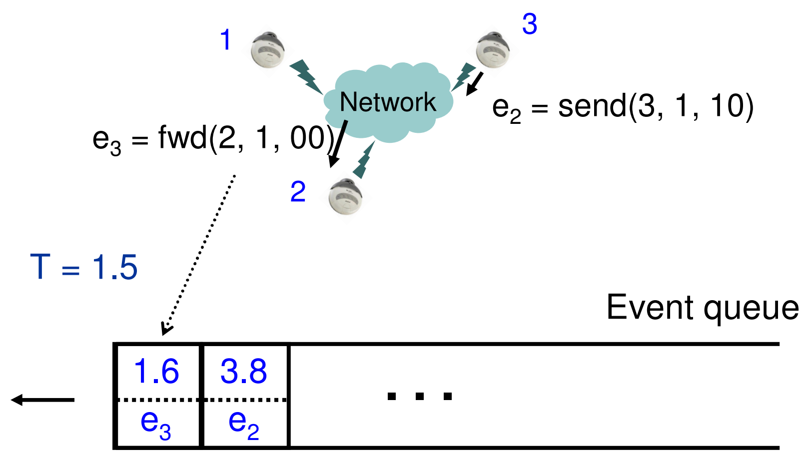
Repeat while there are events in the event queue:

1. Dequeue event at head of queue (“imminent event”)
2. Advance simulation timer to time of imminent event
3. Execute imminent event: update system state
4. Generate future events and enqueue them

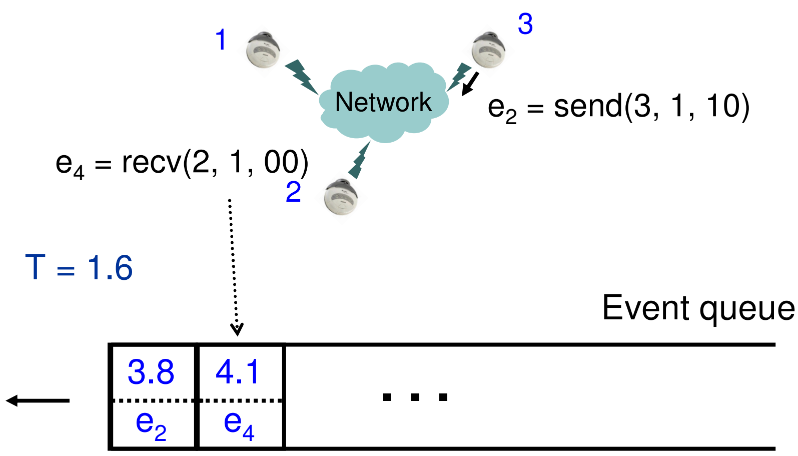
Example of Simulation with Event Queue



Example of Simulation with Event Queue



Example of Simulation with Event Queue



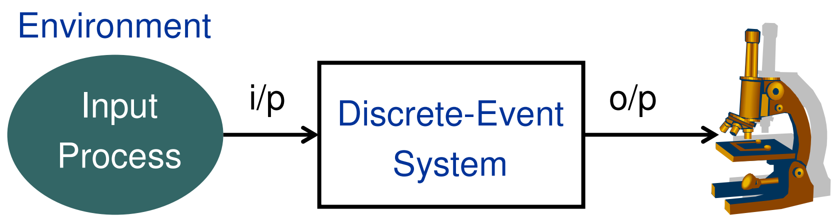
Implementing the Event Queue

- Event with smallest time-stamp must be dequeued
- New events must be inserted into sorted order according to their timestamps

- Efficient Data Structure: Priority Queue

- Particular version: Calendar Queue
  [R. Brown, Comm. of the ACM, 1988, vol. 31(10)]

Simulating a Discrete-Event System (DES)



2.
How to generate random input?

1.
How to simulate the system?

3.
How to check Output?

Generating Random Inputs

Suppose we have an input signal taking values in a set of "events": $\{e_1, e_2, e_3, \ldots, e_n\}$

Suppose event e_i occurs with probability p_i

$\sum_i p_i = 1$

How do we generate events randomly?

Inverse Transform Method

- Generate u in [0,1) – uniformly at random
  - using your programming environment's built-in pseudo-random number generator
  - e.g. drand48() in C
- Add up p_i's until we get to

  $$\sum_{i=1}^{j} p_i \leq u < \sum_{i=1}^{j+1} p_i$$

  Is this an efficient procedure?
- Generate input event e_{j+1}

Analysis of Inverse Transform Method

Inefficient if n is large

Why?

Because we need to compute many partial sums $\sum_i p_i$ in order to figure out where u lies

- worst case: n-1 such sums

There's one special case where sampling from the p_i's is easy: what is it?

Solution: The Accept/Reject Method

Easy to generate events uniformly at random efficiently

But in general, we have an arbitrary probability mass function $p_1, p_2, \ldots, p_n$.

How can we leverage the ease of generating uniformly at random to generate according to the p_i's?

→ The accept/reject method gives us a way to do this

General Setup

Given:

Efficient method for sampling from n events according to p.m.f. $\{q_1, q_2, \ldots q_n\}$

- e.g. the pmf is uniform random, $q_i = 1/n$ for all i

Need:

Efficient method for sampling from same n events but according to non-uniform p.m.f. $\{p_1, p_2, \ldots, p_n\}$

Constraint: $\forall$ i, $q_i > 0$ iff $p_i > 0$

Any ideas on how to do this?

Idea #1 for Accept/Reject Method

Do two steps:

1. Select index i randomly according to easy distribution $\{q_1, q_2, \ldots, q_n\}$
2. Then *accept* the index i with probability p_i
   - generate event e_i
   - otherwise go back to step 1

How do we implement this?

Idea #1 for Accept/Reject Method

Do two steps:

1. Select index i randomly according to easy distribution $\{q_1, q_2, \ldots, q_n\}$
2. Then *accept* the index i with probability p_i
   - generate event e_i

Two questions:

1. Is this correct? YES, if each $q_i = 1/n$
   - Are we really generating e_i according to $p_1, p_2, \ldots, p_n$?
2. Is this efficient?
   - What is the expected #trials before we generate e_i?

On the order of n

Idea #2 for Accept/Reject Method

Do three steps:

1. Initially: select c such that $p_i/q_i \leq c \ \forall \ i$ s.t. $p_i > 0$
2. Select index i randomly according to easy distribution $\{q_1, q_2, \ldots, q_n\}$
3. Then *accept* the index i with probability $p_i/(c\, q_i)$
   - generate event e_i

Idea #2 for Accept/Reject Method

Do three steps:

1. Initially: select c such that $p_i/q_i \leq c \ \forall \ i$ s.t. $p_i > 0$
2. Select index i randomly according to easy distribution $\{q_1, q_2, \ldots, q_n\}$
3. Then *accept* the index i with probability $p_i/(c\, q_i)$
   - generate event e_i

Claim: This method is

- Correct: $\Pr(e_j \text{ is generated}) = p_j$
- Efficient: Expected #trials = c

Accept/Reject works the same way for Continuous Random Variables, too!

Given: How to generate Y with "easy" probability density function (pdf) $f_Y(t)$

Need: To generate X with pdf $f_X(t)$

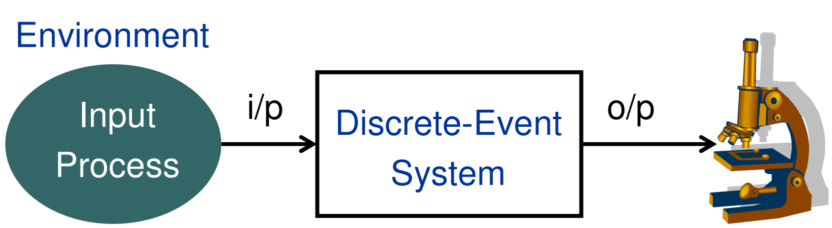
Constraint: $\forall$ t, $f_Y(t) > 0$ iff $f_X(t) > 0$

Algorithm:

1. Pick constant c such that $f_X(t) \leq c\, f_Y(t)$ $\forall$ t s.t. $f_Y(t) > 0$
2. Sample t according to f_Y
3. Accept X = t with probability $f_X(t) / [c\, f_Y(t)]$

Useful for generating inter-arrival times of events

Simulating a Discrete-Event System (DES)



2. How to generate random input?

1. How to simulate the system?

3. How to check Output?

How to check the output

First, we need to know WHAT to check

- i.e., what the system must do

Temporal Logic

A formal notation for specifying what the system must do

- specifies system properties over time

Propositional Temporal Logic

Every send(1, 2) is eventually followed by a recv(2, 1)

G { send(1,2) ⇒ **F** recv(2, 1) }

At any point in time

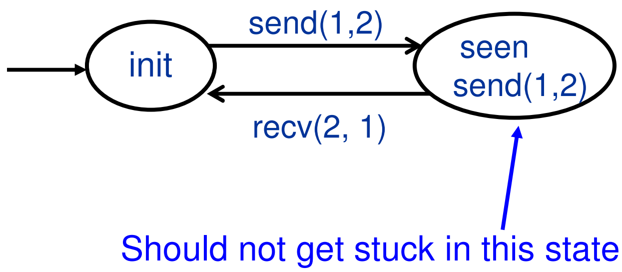
If a send(1, 2) occurs

It is *eventually* followed by a recv(2, 1)

Propositional Temporal Logic

Every send(1, 2) is eventually followed by a recv(2, 1)

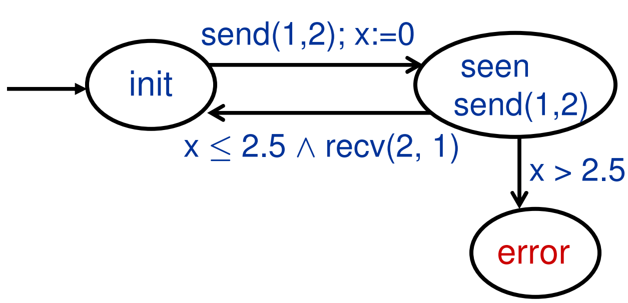
$$\mathbf{G}\ \{\ \text{send}(1,2)\ \Rightarrow\ \mathbf{F}\ \ \text{recv}(2, 1)\ \}$$



Real-Time Temporal Logic

Every send(1, 2) is followed by a recv(2, 1) within 2.5 ms

$$\mathbf{G}\ \{\ \text{send}(1,2)\ \Rightarrow\ \mathbf{F}_{\leq 2.5}\ \ \text{recv}(2, 1)\ \}$$



Next Monday

More on Temporal Logic

Reachability Analysis

- foundation for algorithms for verification and control of discrete and hybrid systems